POLITECNICO DI TORINO

MASTER's Degree in COMPUTER ENGINEERING

MASTER's Degree Thesis

Study of the Future Airborne Capability Environment (FACE) and proposal of an avionics SW architecture implementing Open Architecture (OA), Integrated Modular Avionics (IMA) and Modular Open Systems Approach (MOSA) concepts

Supervisors	Candidate
Prof. MATTEO SONZA REORDA	EDOARDO CAVALLOTTI
Prof. PAOLO MAGGIORE	
Dott. ALESSANDRO PATERNOSTRO	
Ing. ALEKSANDAR PETKOV	

DECEMBER 2024

Study of the Future Airborne Capability Environment (FACE) and proposal of an avionics SW architecture implementing Open Architecture (OA), Integrated Modular Avionics (IMA) and Modular Open Systems Approach (MOSA) concepts

Cavallotti Edoardo

Abstract

Today's military aviation systems are usually developed by a single vendor for specific requirements. While effective, this approach has drawbacks such as long lead times, complex improvement processes, and limited hardware and software reuse across different aircraft, leading to unique designs for each platform.

The complexity of modern mission equipment and electronics has increased costs and integration times, impacting the deployment of new capabilities across the military aviation fleet. Extensive testing and airworthy qualification requirements further exacerbate this issue.

The current procurement system does not encourage hardware and software reuse across programs, and the aviation community lacks sufficient standards for software component reuse. Contributing factors include the small military aviation market, challenges in developing qualified aviation software, and the inability to use commercial software standards due to stringent safety requirements aimed at minimizing risks, mission capability loss, and potential loss of life.

Given all these issues, the military aviation market is evolving towards new paradigms which foster modularity, portability and reusability of its own components. The new picture, show the emergence of new reference architectures with the precise goal of fulfilling the previous requirements, following the concept Integrated Modular Avionics (IMA) and Modular Open System Architecture (MOSA). In this scenario, the new Future Airborne Capability Environment (FACE) Technical standard stands out proposing a solution following strictly the IMA and MOSA approaches. The purpose of this thesis project is to deal with this new standard by means of proposing a valid solution for the problems described before; thus the ultimate goal is proposing an avionic software architecture, conformant with the FACE Technical Standard, in order to foster reusability, interoperability and portability on avionic software. In the first part of this thesis work, a brief presentation about IMA and MOSA topics is given explaining the advantages brought in by these new paradigms and the challenges to face in order to achieve a widespread use of new architectural solutions. After that, FACE fundamental concepts and Share Data Model explanation are provided to introduce the reader to the technicalities of the field and to give her the tools to deal with all the terms and aspects presented in the current document.

In the second part of the thesis, a case study will be discussed in details in order to show a possible fully working scenario of deployment. A premise is need in this

context: the code developed in for this current thesis is a vertical slice of the FACE Standard stack since all the layers are discussed, analyzed and implemented; only few features from each layer, though, are implemented since the Standard provides a very wide set of guidelines which involve disparate aspects, some of which could not be treated in this current work for brevity reasons. The code will realize a FACE Standard data pipeline which starts from the acquisition of aircraft simulated data to the graphical symbology calculation in order to display plane data by means of a window. This pipelines goes through all the FACE layers, especially the PSSS and PCS in which multiple applications reside and communicate to each other by means of the IOSS or the TSS.

This thesis has been conducted inside the company Leonardo s.p.a. - Aircraft Division in Turin. This document is property of Leonardo s.p.a. - Aircraft Division which owns all the rights.

ACKNOWLEDGMENTS

to Jesus which I invoked many times during this work

Table of Contents

List of Figures

List of Tables

Acronyms

AC	Aircraft.
CDM	Conceptual Data Model.
CLI	Command Line Interface.
CPU	Central Processing Unit.
DCPS	Data Centric Publish/Subscribe.
DDS	Data Distribution Service.
DPM	Device Protocol Mediation.
DF	Data Fusion.
EGI	Embedded GPS INS.
FACE	Future Airborne Capability Environment.
GPS	Global Positioning System.
GPU	Graphic Processing Unit.
GUI	Graphical User Interface.
HM	Health Manager.
HMFM	Health Monitoring Fault Management.
HW	Hardware.
HUD	Head-up Display.
ICD	Interface Control Document.
IDL	Interface Definition Language.
IMA	Integrated Modular Avionics *or* Integrated Modular Architecture.
IMU	Inertial Measurement Unit.
INS	Inertial Navigation System.
IO	Input Output.
IOSS	Input Output Service Segment.
IPC	Inter Process Communication.

LDM	Logical Data Model.
LSB	Least Significant Bit.
MW	Middleware.
MOSA	Modular Open System Architecture.
MSB	Most Significant Bit.
MPC	Make Project Creator.
OMG	Object Management Group.
OSS	Operating System Segment.
PCS	Portable Component Segment.
PDM	Platform Data Model.
POSIX	Portable Operating System Interface for Unix.
PSCS	Platform-Specific Common Services.
PSDS	Platform-Specific Device Services.
PSGS	Platform-Specific Graphics Services.
PSSS	Platform-Specific Service Segment.
RTOS	Real Time Operating System.
SW	Software.
TSS	Transport Service Segment.
UART	Universal Asynchronous Receiver-Transmitter.
UML	Unified Modeling Language.
UoC	Unit of Conformance.
UoP	Unit of Portability.
USM	UoP Supplied Model.

Chapter 1

Introduction

The introduction of FACE Technical Standard presents in a very clear manner the current scenario this thesis is starting from, the direction it is moving to and the problems this work aims to solve:

> Today's military aviation community airborne systems are typically developed for a unique set of requirements by a single vendor. This form of development has served the military aviation community well; however, this stovepipe development process has had some undesired side-effects including long lead times, cumbersome improvement processes, lack of hardware and software reuse between various aircraft platforms, which result in a platform-unique design. The advent of significantly complex mission equipment and electronics systems has caused an increase in the cost and schedule to integrate new hardware and software into aircraft systems. This – combined with the extensive testing and airworthy qualification requirements – has begun to affect the ability of the military aviation community to deploy these new capabilities across the military aviation fleet. The current community procurement system does not promote the process of hardware and software reuse across different programs. In addition, the current aviation development community has not created sufficient standards to facilitate the reuse of software components across multiple platforms. Part of the reason for this is the small military aviation market and another part is the difficulty in developing qualified software for aviation. An additional problem is the inability to use current commercial software Common Operating Environment standards because they do not adhere to the stringent safety requirements developed to reduce risk and likelihood of loss of aircraft, reduced mission capability, and ultimately loss of life. [1]

The Future Airborne Capability Environment (FACE) Consortium is a government, industry, and academia collaboration that develops software standards and business approaches for military avionics systems. Managed by the FACE Open Group Consortium, it aims to promote innovation, reduce acquisition costs, and accelerate

software modernization through a Modular Open Systems Approach (MOSA). The FACE Technical Standard provides guidelines for developing and managing avionics software at various levels of complexity and abstraction, facilitating portable capabilities across different services and programs. This open-architecture approach supports increased competition, reduces integration risks, and is increasingly adopted in new defence acquisition programs.

"A Modular Open Systems Approach (MOSA) is a business and technical strategy for designing an affordable and adaptable system" [2]. This approach breaks down large entities into smaller modules with standardized interfaces, and contains a system architecture that allows adding, removing, or replacing individual components without significantly affecting others. "The fives Principles of MOSA are to Establish Enabling Environment, Employ Modular Design, Designate Key Interfaces, Select Open Standards, Certify Conformance" [2].

This thesis work is divided into the following sections, each of them aiming to provide an analysis or an explanation of the context, a proposal of solution in the form of a wide general example or case study, and a conclusion.

- Background: in this section, after a brief introduction into the concepts of Integrated Modular Avionic (IMA) and Modular Open System Architecture (MOSA) in order to familiarize with the new emerging scenario, a commented and explained summary of the FACE Technical Standard and Open DDS is provided reporting the key points from the documentation in order to give all the necessary knowledge to understand the content of the current thesis.

- Proposed Solution: this chapter consists of a proposal for an architecture realizing the paradigms of Open Architecture, IMA and MOSA; both conceptual high level solution and low code level solution are provided and explained. The organization of the abstract entities is provided along with the organization of the software components.

- Implementation Details: the design of the Software Architecture is discussed in details. Each SW application is presented and analysed.

- Achieved Results: a discussion about what has been done, with particular interest in the limitations and strength points, is provided here.

- Conclusions: an analysis of all the work that has been done with a particular focus on advantages and disadvantages introduced by the adopted solution with respect to a standard solution.

The case study faced in this thesis is the development of a fully functional vertical section of the FACE layers which aims to give a demonstration of the capabilities of this new standard, investigating the difficulties and the problems which can arise in the adoption of this standard. The software for the thesis comprises several applications, spread all over the FACE Segments which, thanks to the adoption of the Standard, are easily reusable and portable. In addition, a graphical interface

allows the user to interact with a very simplified model of an aircraft with the very purpose of generating pseudo realistic data to feed into the FACE data pipeline. The aim of the simulator is only to forge data for the FACE architecture and check the correct functioning and communication of the applications inside the FACE boundary. Inside the simulator, two emulated sensors (EGI and GPS) provide data to a bottom up pipeline which goes from the IOSS to the PCS and comes back down the architecture to be displayed by the PSSS. In other words, once data are produced, they are sent - in different format - to two applications which are responsible of reading the data. These apps then pass the new info to an another software which combines information, removes noise and sends the filtered data to a display which shows aircraft position, speed and attitude to a user. Beside this pipeline, a health monitoring application is run to check the status of the several software components and, in case of need, to kill and rerun the faulty application. The communication between all the components is allowed by two approaches, serial bus and Data Distribution Service: the former is a very base and well-known technology while the latter is a very interesting and promising paradigm which deserves a deeper investigation and discussion. The use of the FACE guidelines in code development allows the adoption (and the writing) of a wide raster of APIs which favours the portability of the software; indeed, the points of contact and data exchange, between software segments, e.g. the several executables, are permitted by using well-known function signatures and returned values: as long as interfaces between software component are written using these known APIs, communication is straightforward since the developer knows which functions is allowed to use and which data to expect. The use of the FACE guidelines guarantees also modularity of the software segments in which each software component is independent from the others, and no knowledge of the internal structure is needed when integrating them, but only the way they communicate and the data they exchange. This favours the possibility to change an application which receives a specific type of data and elaborates it in a certain manner with another one which receives the same data across the same media but does a completely different job: no extra work for integration is needed since the applications, compliant with the FACE standard, are perfectly compatible thanks to a common interface.

Chapter 2

Background

2.1 Necessity for emerging philosophies

Until most recent times, military aviation systems have been typically designed to meet specific requirements by individual vendors, leading to platform-unique designs. This traditional approach, characterized by closed systems almost monolithic in their design, has caused challenges, including long lead times, difficulty in implementing improvements, and limited hardware and software reuse. As mission systems grow more complex, integration costs and timelines increase, affecting fleet-wide deployment of new capabilities. The lack of hardware and software reuse standards and reliance on unique designs arises partly from the small market and the difficulty of creating qualified aviation software. In this problematic scenario, in the recent past, the need for a transition towards a new paradigm has become increasingly urgent. There was a necessity for a renewal in the conception of avionics systems and this brought to Integrated Modular Avionics [3, 4], Modular Open System Approach [2, 5] and Open System Architecture new philosophies for the military aviation market promoting the adoption of open standards and interoperability between components from different suppliers. MOSA and OSA have made it possible to standardize interfaces and protocols, facilitating the integration of advanced technologies into on-board systems more quickly and economically. This transition made the aircraft more versatile and sustainable, ensuring that they could be upgraded with less logistical and operational impact, thus better adapting to future needs and significantly improving the efficiency and effectiveness of missions.

Thus, to reformulate what has just been said, the real-time Aerospace and Defence applications must employ a MOSA and OSA approach to:

- Minimize integration, maintenance and upgrade costs
- Foster reuse
- Enable rapid reconfiguration to meet evolving mission requirements.

Consequential to this topic, since architecture are becoming less centralized and composed of several modules which need to communicate to each other with real-time

and reliability constraints, new data distribution middlewares are emerging bringing in innovation and new approaches. Among many, with its interoperability, portability, loose coupling and real-time Quality of Service (QoS), the Data Distribution Service (DDS™) is showing its potential in becoming a standard for implementing mission-critical open architecture systems, remaining for now in careful evaluation.

2.2 A brief mention of MOSA

A Modular Open Systems Approach (MOSA) [5] is a comprehensive strategy aimed at achieving cost-effective development, acquisition, and maintenance over a system's life cycle. This approach emphasizes the use of modular designs with interfaces built on widely accepted standards, allowing for straightforward verification of compatibility and functionality. MOSA is widely applicable across various sectors and encourages a modular architecture and an open business model to facilitate the addition, modification, replacement, or removal of system components throughout the acquisition process. By implementing MOSA, organizations can design systems with loosely coupled, highly cohesive modules, allowing for the independent selection and integration of components from multiple suppliers. This modularity enables quicker integration of emerging technologies and greater adaptability to shifting requirements, ensuring systems can evolve alongside technological advancements. A critical factor in MOSA's success is its reliance on an open business model, which fosters transparency and collaboration across participants. This model facilitates shared risk, encourages asset reuse, and reduces overall ownership costs, while enhancing flexibility, competition, and innovation. Through MOSA, organizations can incrementally build and enhance capabilities – whether systems, subsystems, software, or services – while keeping pace with the rapidly evolving technology landscape.

2.3 Integrated Modular Avionics

An Integrated Modular Avionics (IMA) [3, 4] is a modular open standard computing platform which provides general processing capability and finds large use in the civil transport aviation; since the recent past, this paradigm has been employed also in the military aircraft sector. A progressive transition from Federated Architecture to IMA paradigm is happening involving several markets, not only aviation but also automotive where the ECU (engine control unit) are getting redesigned in the context of IMA. Federated Architecture, which is leaving place to IMA, is a pattern in enterprise architecture that allows interoperability and information sharing between semi-autonomous decentralized systems. Compared to the Integrated Modular Avionics (IMA) paradigm, federated architecture (FA) presents several disadvantages such as duplication of resources across semi-autonomous system leading to reduced efficiency, reduced synchronization and consistency of data, higher maintenance complexity and expense, lower robustness, security and scalability. In summary, FA

main drawbacks with respect to IMA include reduced efficiency, greater management complexity, less integrated security, and more challenging maintenance and upgrades—factors that can impact performance, particularly in critical environments like avionics. Thus, IMA aims at mitigating these problems by increasing modularity, reusability and interoperability. In Figure 2.1 it's possible to compare the two paradigms and understand the various advantages introduced by the IMA approach.

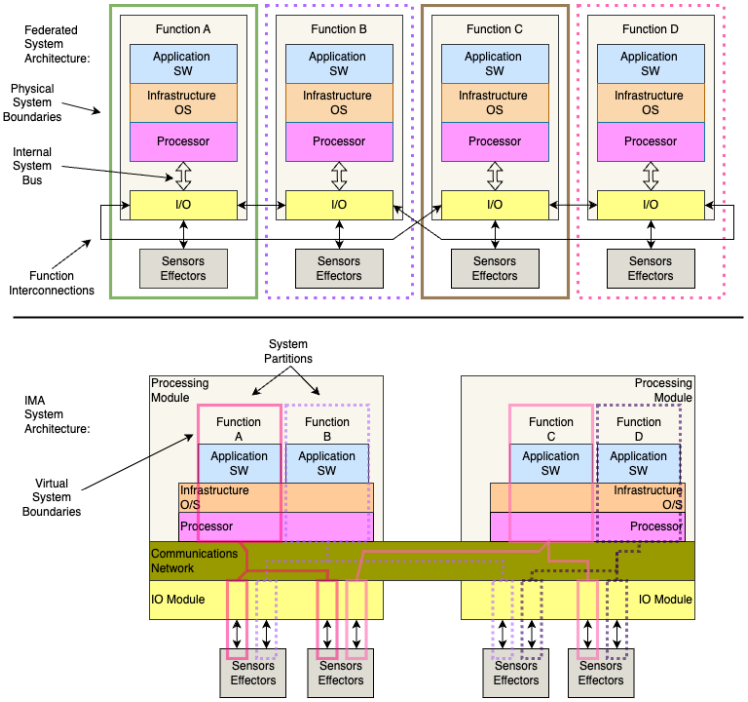

Figure 2.1: IMA vs Federated Architecture [6]

2.3.1 IMA Paradigm

The modular design of an IMA provides a number of advantages such as cost, space and power savings, shorter development time and higher growth potential. These advantages are almost the same the ones FACE aims to provide since this Standard is moving in this very direction with the very same goals in the field of military aviation. Indeed, the modular aspect of IMA paradigm allows to rethink systems as a set of components mutually integrated which communicate to each other with open standards and each component can be replaced or upgraded without changing any of the other parts [4, 3]. This aspect allows the military to make use of civil aircraft technologies and applications adding only specific military components without having to develop the entire system by scratch. In order to achieve this goal, Open Standards are used in every aspect and in every module of the IMA system, including IMA Network, IMA Modules and IMA Hardware.

- The IMA Network should provide a common method to access and exchange data. Thus, the same protocols are to be used inside and between modules and computation platforms, such as Ethernet.

- The IMA Hardware should use specifications and technologies which are freely distributed as open (source) standards such as PCI.
- IMA Software should be based on freely available software development environment and large supported APIs such as POSIX APIs or ARINC 653.

Through the use of open standards there is no vendor lock-in (monopoly) and components can be upgraded or substituted in a very quick and efficient manner. Figure 2.1 shows the difference between an IMA configuration and a Federated Architecture realization: the aforementioned advantages offered by the IMA with respect to FA are clearly visible in the picture.

2.3.2 LRU

In the context of IMA, an important keyword is LRU which stands for Line Replaceable Unit and consists of a computing platform which the system is composed of. Every function of a system is implemented in one or more separate LRUs and each LRU has its own power supply, computing resources and interfaces. In a federated system the integration of LRUs is made by point to point wiring increasing the complexity. In a IMA approach, there are no more separated LRUs but only "Virtual LRUs" since the applications and software share common General Processing Modules, OS Infrastructure and IO, in a virtually partitioned environment using technologies as ARINC 653.

2.3.3 Advantages of IMA

The first advantage of IMA is represented by the reduced number of LRUs since the functions of multiple units can be combined and implemented on a single processing unit which hosts different applications. Another important feature is the modular design which aims to improve growth and maintainability of the system by using IMA modules in a minimum number of LRUs. The modularity is enabled by the use of interfaces which allows different modules to communicate in a common way.

2.3.4 OSA

The Open System Architecture shares several commonalities with the IMA paradigm. This concept is the derivation of the modular open system approach (MOSA) and the direction is the one of designing an architecture that supports affordable change, enables evolutionary acquisition and spiral development as is enabled and encouraged by the search for portability and reusability. OSA is both a business and technological strategy for developing a new system or modernizing an existing one, through modular design, consensus-based standards and open technologies. This is pretty much the same IMA aims to, thus OSA and IMA can be seen almost as same concept.

2.3.5 IMA vs OSA

OSA is mainly driven by the need to accommodate new future mission capabilities that may not be fully defined during vehicle development. OSA allows for flexibility to add or adjust capabilities as needed. In contrast, the IMA architecture prioritizes aircraft weight reduction, simpler system installation through minimized wiring and physical units, and reduced change costs both during initial development and future updates. Interestingly, both OSA and IMA offer similar advantages, though they stem from different priorities. However, IMA places a stronger emphasis on safety and certification compared to the mission-oriented focus of OSA [3]. So, in summary, IMA focuses more on the integration of hardware components aiming at weight reduction and installation simplification. OSA, on the other hand, aims at offering a flexible software structure which fosters new capabilities integration.

2.3.6 Information Assurance

According to IMA, a military system will use civil applications to control the aircraft and military specific applications to carry out missions. This approach comes with the necessity of keeping separated classified and unclassified information which otherwise would be on the same platform since the coexistence of military and non military technologies. The term Information Assurance, indeed, means the prevention of unauthorized disclosure of classified data to unclassified systems while unclassified civil applications must be able to be hosted alongside classified military application sharing certain computing resources but keeping separated. Three technologies are available to guarantee information assurance.

- Data Separation → This technique implies keeping classified data separated from unsafe systems: this can be done at a system level by duplicating hardware and keeping the two parts on different devices, at module level or at application level. This last method is the most efficient since no duplication of hardware is required but additional isolation techniques are to be applied.
- Encryption → Encryption can be used in data communication and data storage. By means of encrypting information, data can be stored in a secure way on unsecure device. This can be done either in software or in hardware. In an analogue way, encrypted communication allows secure data to be passed along an unsecure channel.
- Zeroization → This techniques aims to render classified information useless by writing data patterns to memory several times in order to corrupt data and make them unreadable.

The emergence of the network vulnerabilities in cloud connected aircraft will also require new refinement in criteria for IMA cyber security.

2.3.7 Emerging Architecture Challenges

The first necessity is to establish some mainstreams data models and software frameworks to enable software reuse across avionics platforms. The emergence of FACE offers a step in the right direction since it provides a data model and an architecture for next generation mission, flight and autonomy with regard to application software units of portability. Alongside, there is a need for improvement in avionics security and safety in these new environments and, again, FACE moves in this direction by applying the concept of Profiles.

2.4 DDS

The DDS specification outlines a Data-Centric Publish-Subscribe (DCPS) model for communication and integration in distributed applications. It defines the Application Programming Interfaces (APIs) and Communication Semantics (behaviour and Quality of Service) required for efficient information delivery from data producers to corresponding consumers. The goal of the DDS approach can be summarized as enabling the "Efficient and Robust Delivery of the Right Information to the Right Place at the Right Time"[7].

The Data Distribution Service (DDS) for real-time systems is an Object Management Group (OMG) machine-to-machine (sometimes called middleware or connectivity framework) standard that aims to enable dependable, high-performance, interoperable, real-time, scalable data exchange using a publish–subscribe pattern [8]. DDS is the only open standard for messaging that supports the unique needs of both enterprise and real-time systems. The open interfaces and advanced integration capabilities of DDS slash costs across a system's lifecycle, from initial development and integration through ongoing maintenance and upgrades.

DDS fosters the development of loosely coupled, modular and open architecture systems. By supporting well-defined interfaces between components and subsystems, DDS eradicates stovepipe, closed and proprietary architectures. This eliminates complexity to reduce integration, maintenance and upgrade costs, promotes competition at the subsystem and middleware levels and eases reuse. It's evident how this standard meets the modularity and interoperability requirements of IMA and MOSA making the DDS a valuable candidate for transport level middleware. DDS standardizes powerful messaging semantics that reduce development and integration costs while improving system scalability and robustness. Across many industries, DDS is the chosen or required connectivity standard for mission- and safety-critical applications.

DDS is structured to clearly separate the publishing and subscribing components. An application that functions solely as a publisher can include only the elements directly related to publication, while an application that acts only as a subscriber can incorporate only the components specific to subscription.

Quality of Service (QoS) is a concept used to define the expected behaviour of a

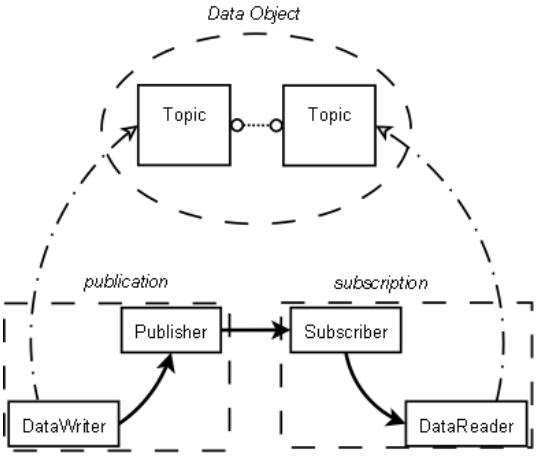

Figure 2.2: DDS Data Flow through components [9]

service. By configuring QoS parameters, developers can specify what outcomes are desired rather than detailing how those outcomes should be attained. Typically, QoS consists of multiple policies, each of which independently associates a descriptive name with a defined value. Defining QoS through a set of distinct policies provides greater flexibility in tailoring service behaviour [7]. In other words, the term QoS refers to a rich set of characteristics that define the behaviour of the DDS systems, among which the most common are *reliability* (determines whether the service is allowed to drop samples), *liveliness* (checks to make sure expected entities in the system are still alive), *durability* (keeps trace of messages sent before the subscriber was alive), *history* (controls what happens to an instance whose value changes before it is communicated to all subscribers) and *resource limits* (controls resources that the service can use to meet other QoS requirements).

2.4.1 OpenDDS

OpenDDS is an open source C++ implementation of the Object Management Group (OMG) Data Distribution Service (DDS) [8, 10] which permits to exchange data in a distributed system like the one under examination. OpenDDS is designed to be "data-centric," meaning it puts the data itself rather than implementation details at the center of communication. Topics are defined based on the data they carry, and communication occurs transparently with respect to such data. OpenDDS complies with the DDS specification, ensuring interoperability with other compliant DDS systems: this means that applications developed with OpenDDS can communicate with other DDS applications even if they use different implementations. For what regards the requirement of a Configurable QoS, OpenDDS provides a wide range of options for configuring the Quality of Service (QoS) of communication. This includes reliability policies, re-transmission times, priority management, bandwidth management and more, allowing developers to tailor communication to specific application needs. OpenDDS implements DDS policies by means of DCPS (Data-Centric Publish-Subscribe): an architecture and set of specifications designed to

facilitate asynchronous, distributed communication between software components within a distributed system. This architecture is based on the publish-subscribe communication model, where components (called publishers) can publish data on specific topics, and other components (subscribers) can subscribe to receive those data. DCPS can be seen as a library or middleware for the DDS paradigm.

2.4.1.1 Key Points of OpenDDS

The main strengths of OpenDDS are [10]:

- Data-Centric Publish-Subscribe (DCPS) Model: implements an efficient publish-subscribe model ideal for distributed data-centric applications, with flexible QoS policies.
- Pluggable Transport Architecture: supports various transport mechanisms (TCP, UDP, multicast, shared memory) that can be configured without modifying application code.
- Flexible Configuration: allows detailed settings via configuration files, managing transport, debugging, memory allocation, and more for publishers and subscribers.
- Centralized DCPSInfoRepo: uses a central repository to manage publisher-subscriber associations, simplifying communication in distributed environments.
- Comprehensive Quality of Service (QoS) Support: offers a range of QoS policies (e.g., reliability, resource limits, history) to meet application requirements.

2.4.1.2 Publish & Subscribe

DDS is a publish and subscribe service. Data values (samples of instances for a topic) are transferred through the system for conceptual *Data Objects*. The *publication* (the association of a Publisher and a DataWriter) sends Samples to one or more *subscriptions* (the association of a DataReader and a Subscriber).

2.4.1.3 Basic Components of the System

As shown in Figure 2.2 the basic components are Topic, Publisher, Subscriber, DataWriter, and DataReader.

- Topics are information about a single data type and the distribution and availability of samples.
- Publishers apply control and restrictions to data flow from DataWriters.
- Subscribers apply control and restrictions to flow of data from DataReaders.
- DataWriters create Samples of a single application data type.

- DataReaders receive Samples of a single application data type.
- A Publisher can have many DataWriters.
- A Subscriber can have many DataReaders.
- A DataWriter has a single Topic.
- A DataReader has a single Topic.
- A Topic can have many DataReaders and DataWriters.
- A "publication" can have many associated "subscriptions".
- A "subscription" can have many associated "publications".

2.4.1.4 Data Flow

The application on the publishing side initiates the flow of data by writing a data value to the DataWriter. The DataWriter's publication publishes the Samples and the action of publication sends the Samples to the associated subscription(s). Each associated Subscriber gives the received Sample to its DataReader(s) that are associated with the sending DataWriter. The flow ends with the application on the subscribing side retrieving the data from the DataReader.

Quality of Service (QoS) Policies control the flow of the data through the system. The Topic, DataReader, DataWriter, Publisher, and Subscriber all have QoS polices. The QoS policies of Publisher, DataWriter, and Topic control the data on the sending side. QoS policies of Subscriber, DataReader, and Topic control the data on the receiving side.

The *Data Objects* are identified by Topics. Topics are compatible when they have the same name, the same data type and the QoS polices are not in conflict. When a DataReader's Topic is compatible with a DataWriter's Topic, then the *publication* and *subscription* become associated and data is published between them.

The DCPSInfoRepo is the OpenDDS object that maintains the state of the Domain(s): it is a side process which supervises the exchange of information and collects a log of what happens over the communication. The DCPSInfoRepo detects when subscriptions and publications in a domain should be associated and notifies them to make the associations.

2.4.1.5 Transport

OpenDDS supports *one to one* (point to point) and *one to many* (multicast) styles of publishing. OpenDDS separates the transport from the higher level protocols by means of an Extensible Transport Framework (ETF) which is a layer above the transport protocol, independent from it, which allows the information exchange amongst Publishers and Subscribers. Several transport protocols can be used: TCP is reliable, RTPS/UDP is the preferable one while UDP must be avoided since it doesn't support reliability at all. Multicast is another option via UDP.

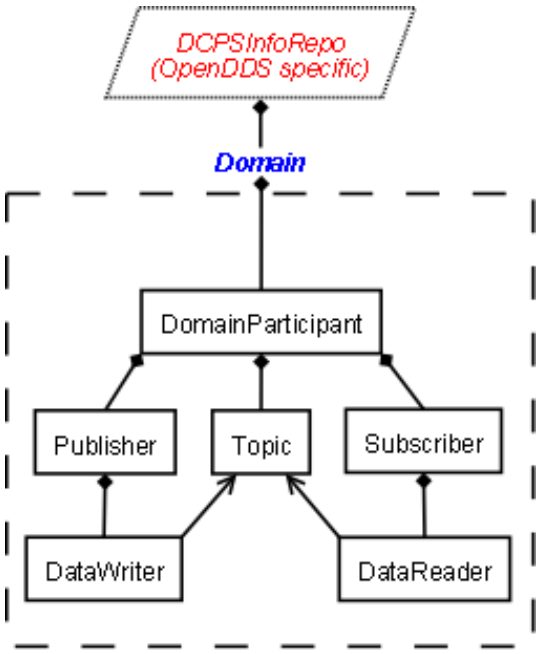

Figure 2.3: DDS Fundamental Components [9]

2.5 FACE Architecture

The Open Group [11] Technical Standard for the Future Airborne Capability Environment (FACE™), embodies a set of requirements and descriptions referred to as the FACE Standard [1]. The FACE Standard defines the software computing environment and interfaces designed to support the development of portable components across the FACE general-purpose, safety, and security profiles. FACE uses industry standards for distributed communications, programming languages, graphics, operating systems, and other areas as appropriate. The goal of FACE is to reduce software development and integration costs and reduce time to field new avionics capabilities. FACE establishes a common computing software infrastructure supporting portable, capability-specific software components across avionics systems. As presented in the FACE Technical Standard document, the FACE technical approach tackles barriers to modularity, portability, and interoperability by defining a Reference Architecture and employing design principles to enhance software portability. To meet the objectives of the FACE Technical Standard, several software engineering practices have been employed focusing on the following principles:

- Use published industry standards to provide normative references, allowing the use of existing software libraries and tools whenever possible.
- Use profiles to define subsets of those standards when support of the entire standard would lead to safety or security certification issues, or when supporting only a defined subset would lead to a more cost effective solution; a profile can also reference a specific version of a standard in its entirety.
- Use a standardized architecture describing a conceptual breakdown of function-

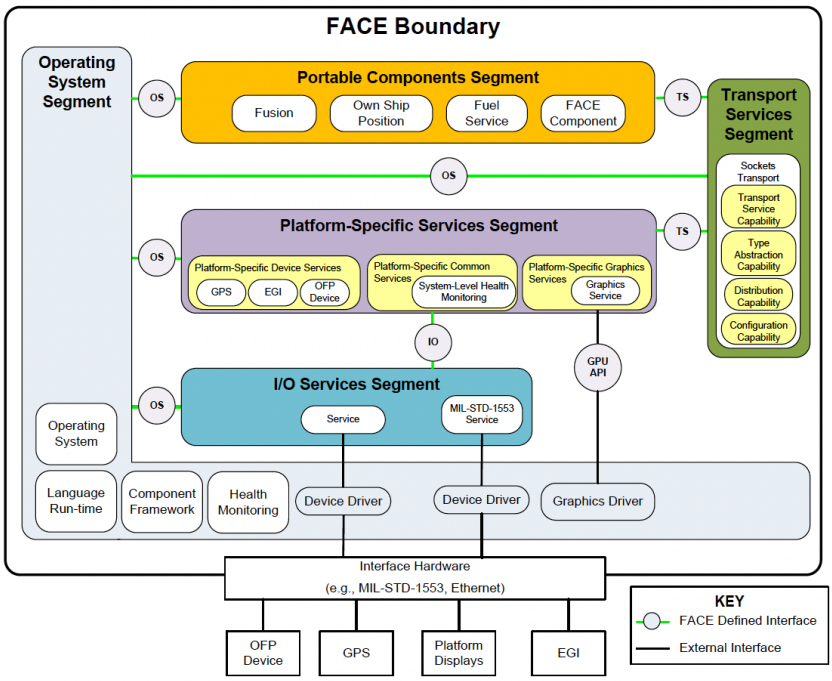

Figure 2.4: FACE Architecture [1, p. 16]

alities and the FACE Reference Architecture to promote the reuse of software components to share common functionalities across military systems (see Figure 2.4).

- Define standardized interfaces to allow software components to be moved between systems developed by different vendors.
- Use a data architecture to ensure the data communicated between the software components is fully described to facilitate the integration on new systems.
- Require that hardware abstraction be used to decouple software components from specific hardware implementations, and device driver normalization be used to allow interfaces to external devices to be developed independently of the computing platform device drivers.
- Use a display window management strategy to incorporate common avionics user interface standards to aid in the integration of components needing to share display areas and input devices.

This technical approach enables software components to be redeployed on other platforms to achieve greater portability and interoperability when standardized FACE Interfaces are used.

2.5.1 Open Group

"The Open Group is a global consortium that enables the achievement of business objectives through technology standards. The diverse membership of more than

925 organizations includes customers, systems and solutions suppliers, tool vendors, integrators, academics, and consultants across multiple industries" [11].

The Open Group offers an effective methodology to form and govern standards initiatives and to develop standards and certification programs. With decades of experience and a proven track record of launching and managing successful certifications and standards programs, The Open Group's experts tailor the process to each initiative's specific needs. The internationally recognized process provides a commercially and legally sound foundation for the formation and governance of standards initiatives and special interest groups that accelerates the path to success.

The Open Group members lead the development of technology standards and thanks to the collaboration with customers, suppliers and other organizations, they are able to capture, clarify and integrate current and emerging requirements, establish standards and policies, and share best practices. These standards ensure openness, interoperability and consensus.

The Open Group Standards Process defines the procedures for key tasks with The Open Group Standards Development activities, in order to visualize and make into standards company requirements.

Standards Process is made up of:

- Purpose
- Principles
- Standards Development Process (Core and Supporting Processes)

2.5.2 UoC and UoP

Throughout all the current document, many acronyms related to multiple concepts have been used and all are reported inside the Glossary at the beginning of the thesis. Some of them will be explained and deeply analysed moving forward in the text but two of them are to be clarified right here since are a sort of prerequisite for the further comprehension.

The first is *Unit of Conformance, UoC* which is "A software component or domain-specific data model designed to meet the applicable requirements defined in the FACE Technical Standard. It is referenced as a UoC at any point in its development, and becomes a FACE Certified UoC upon completion of the FACE Conformance Process" [1, p. 10].

The second is *Unit of Portability, UoP* whose definition is "A UoC that resides within a PCS or PSSS" [1, p. 10].

2.5.3 PCS

The Portable Component Segment is comprised of software components providing capabilities and/or business logic. PCS components are intended to remain agnostic from hardware and sensors. Additionally, these components are not tied to any data

transport or operating system implementations to meet objectives of portability and interoperability.

The Portable Component Segment is represented by all those software components which can be deployed on different systems without reconfiguration or rebuilding of the application itself. In this segment different apps can find place and all the portable software is allocated here. This layer is the fundamental key point of the FACE architecture.

2.5.4 TSS

The Transport Service Segment is comprised of communication services. The TSS abstracts transport mechanisms and data access from software components facilitating integration into disparate architectures and platforms using different transports. TSS UoCs are responsible for data distribution between PCS and/or PSSS UoCs. TSS capabilities include, but are not limited to, distribution and routing, prioritization, addressability, association, abstraction, transformation, and component state persistence of software component interface information.

The Transport Segment, from which derives the TSS for Transport Segment Services, is a transport layer which allows the communication between PCS and PSSS, carrying messages in a publisher/subscriber or request/reply fashion by means of several transport protocols. In the current case the implementation of the TS is done by means of DDS by OpenDDS (2.4 using RTPS (Real Time Publish Subscribe Protocol) with both UDP and TCP. RTPS is now known as DDS Interoperability Protocol (DDSI). This wire protocol uses the OMG Common Data Representation (CDR) to encode data in a platform-neutral way such that it can be sent over the network.

2.5.5 PSSS

The PSSS is comprised of sub-segments including Platform-Specific Device Services, Platform-Specific Common Services, and Platform-Specific Graphics Services.

- Platform-Specific Device Services (PSDS) are where management of data and translation between platform-unique Interface Control Documents (ICDs) and FACE UoP Supplied Models (USMs) occurs.
- Platform-Specific Common Services (PSCS) are comprised of higher-level services including Logging Services, Device Protocol Mediation (DPM) Services, Streaming Media, Health Monitoring and Fault Management (HMFM), and Configuration Services.
- Platform-Specific Graphics Services (PSGS) is where presentation management occurs. PSGS abstracts the interface specifics of Graphics Processing Units (GPU) and other graphics devices from software components within the FACE Reference Architecture.

According to FACE Software Architecture 2.5, inside the PSDS subsegment, applications to manage sensors hardware find place, examples are the apps to manage data from EGI and GPS. Inside the PSCS an important role is played by the DPM component. The aforementioned apps and components are explained in the following paragraphs.

2.5.5.1 Embedded GPS INS

The Embedded GPS INS (EGI) is a sensor system designed to measure motion. It integrates a GPS (Global Positioning System) for determining position and an INS (Inertial Navigation System) for measuring accelerations and angular velocities relative to three coordinate axes. The accelerations recorded by the INS are processed and integrated to derive velocity and position estimates. However, these computed values are prone to drift, meaning the errors accumulate over time. To mitigate this drift and ensure the accuracy of velocity and position measurements, a Kalman Filter is employed to correct the integrated results using GPS data. [12]

2.5.5.2 Global Positioning System

The Global Positioning System (GPS), formerly known as Navstar GPS, is a satellite-based radio navigation system owned by the United States Space Force and operated by Mission Delta 31. It is one of several global navigation satellite systems (GNSS) that provide geolocation and time information to GPS receivers anywhere on or near Earth, as long as there is an unobstructed line of sight to at least four GPS satellites. GPS operates independently of the need to transmit data from the user and does not rely on telephone or Internet connectivity, though these technologies can enhance its functionality. It delivers essential positioning services to military, civilian, and commercial users worldwide. While the GPS system is developed, maintained, and controlled by the U.S. government, it is freely available to anyone with a GPS receiver. [13]

2.5.5.3 Device Protocol Mediator

The DPM is a set of UoCs in PSCS whose task is to act as protocol mediator for platform devices using transport protocols (e.g., SNMP, SNMP V3, HTTP, HTTPS, FTP and so on). DPM Services communicate with PSDS sub-segment UoCs through the IOS Interface. For example, if a certain UoC, in PSDS on a certain device platform, wants to use a protocol instead of another one, it shall write to DPM the original protocol data and retrieve the new protocol data. The IO connection created to allow communication between DPM and the PSDS UoC can be realized by means of sockets.

2.5.6 IOSS

The I/O Services allow the communication with external and peripheral devices. This segment permits to establish connections not only with the outside of the computational platform but also between components inside the same segment when this type of communications is preferred against TS. The strength points for the I/O Service are clarity, consistency and extensibility. Clarity is provided by defining a separate I/O Service for each supported I/O bus architecture. Consistency is provided by supporting multiple I/O bus architectures that follow a consistent declaration pattern for an I/O Service. Extensibility is addressed by means of the capability of extending the configuration and controlling declarations for an I/O Service. Each interface follows a consistent pattern to be used with the same syntax for common functions. An instance of an I/O Service represents a particular device bus in the system and the type of this instance is the device bus architecture under consideration.

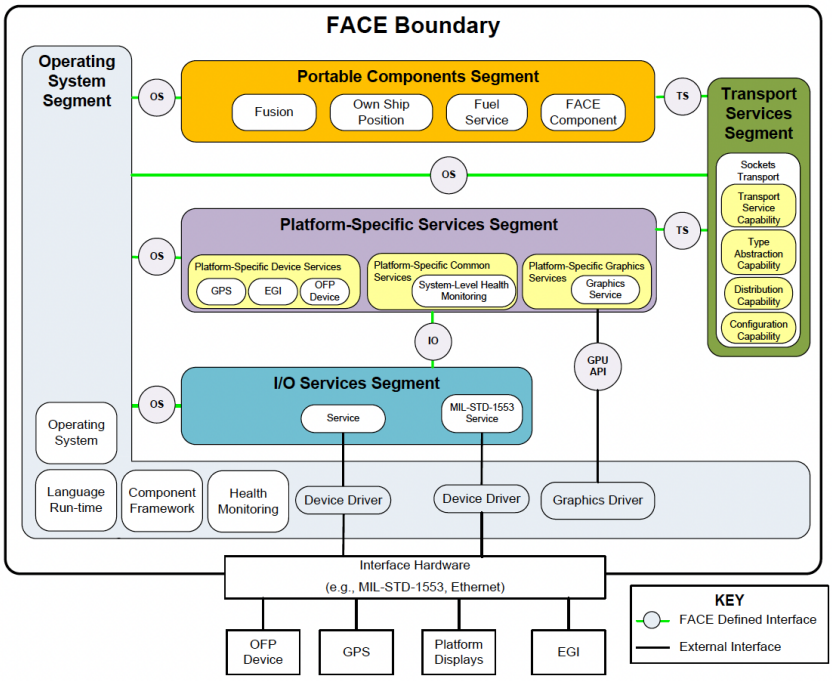

Figure 2.5: FACE reference architecture [1, p. 16]

2.6 FACE Operating System Segments Profiles

2.6.1 Introduction to Profiles

Since the FACE conformant UoCs can be installed in different systems, a different level of safety, security and airworthiness must be achieved accordingly to three FACE profiles [1, p. 18]. The FACE Reference Architecture define three OSS Profiles (Security, Safety and General Purpose) tailoring Operating System (OS) APIs, programming languages, programming language features, run-times, frameworks and graphics capabilities to meet requirements in terms of levels of criticality. The

FACE Profiles are defined in the FACE Technical Standard primarily in terms of API methods available per profile; the same applies on constants and if supporting APIs are not in a profile, then related constants should not be either. The impact of applying the FACE profiles into the code is represented by the set of APIs the software component can use or not, with consequent impacts to the algorithm. An example could be the set of APIs devoted to the management of directories described inside the Standard as *POSIX_FILE_SYSTEM*: these syscalls are excluded from the Security profile and in some cases are relegated to the General Purpose only. ([1, p. 185])

DO-178C [14] is a fundamental document about the airworthiness and validity of avionic code: indeed, the purpose of this document is to provide guidance for the production of software for airborne systems and equipment that performs its intended function with a level of confidence in safety that complies with airworthiness requirements. The FACE Technical Standard is thought to match exactly this document and to not create any overlapping or conflict. FACE's division in profiles is designed to support hazard and safety development, in fact, just as example, Safety Base and Safety Extended sub-profiles are designed to support implementations which may have safety impacts. These concerns have been taken into account by the FACE Technical Standard in order to deliver the most safe and secure software possible in accordance with airworthiness levels.

The FACE Conformance process is composed of three stages, which the software component must undergo through:

- FACE Verification: verification whether the software component meets all the requirements or not.
- FACE Certification: the software component is guaranteed as package level.
- FACE Registration: the software component is inserted into the FACE Library.

The same process con be seen from a different perspective, the one of the different parties which validate the software component:

- FACE Certification Authority
- FACE Library Advisor
- FACE Trademark Licensor

2.6.2 Security

The Security Profile constrains the OS APIs to a minimal useful set allowing assessment for high-assurance security functions executing in a single address space (e.g., a POSIX process or ARINC 653 partition). The Security Profile requires ARINC 653 support.

2.6.3 Safety

The Safety Profile is less restrictive than the Security Profile and constrains the OS APIs to those that have a safety certification pedigree. The Safety Profile is made up of two sub-profiles: Safety-Based and Safety-Extended. The Safety Base Sub-profile supports single POSIX process applications with a broader OS API set than the Security Profile. The Safety Extended Sub-profile includes all of the Safety Base Sub-profile OS APIs along with additional OS APIs and optional support for multiple POSIX processes. The Safety Profile requires ARINC 653 support.

2.6.4 General Purpose

The General Purpose Profile is the least restrictive and least constrained and allows the use of the widest set of APIs meeting real-time deterministic or non-real-time non-deterministic requirements.

2.6.5 POSIX vs ARINC 653

ARINC 653 is an avionic standard which offers a set of APIs called APEX (APplication EXecutive) for process management and port communication. A very important note is that ARINC 653 uses a different nomenclature with respect to POSIX in terms of processes and threads; indeed, what POSIX calls either process or thread ARINC 653 calls it process, making no distinction, but both cases are executed inside a partition, an isolated and protected environment (both in terms of time and space) that can host multiple processes. For example, the POSIX syscall *pthread_create()* is mapped to ARINC 653 *CREATE_PROCESS()*. Since ARINC 653 requires a specific Operating System, the implemented frame doesn't use this standard and is based on POSIX only. It should be noted that there are important limitations/restrictions on compatibility between POSIX and ARINC 653. For example, only a subset of POSIX APIs (such as those for file or signal management) are supported, as many POSIX features (such as thread management and shared memory) may violate the hard partitioning model of ARINC 653.

2.7 FACE Data Architecture

The FACE Data Architecture [15] defines the FACE Data Model Language (including the language binding specification), Query and Template language, FACE Shared Data Model (SDM) and the rules of construction of the Unit of Portability (UoP) Supplied Model (USM). A clarification must be made before continuing: each FACE-conformant software component has the name of UoC (Unit of Conformance); the UoCs residing in the PCS or in PSSS take the name of UoP (Unit of Portability). Each PCS UoC, PSSS UoC, or TSS UoC providing TS Interfaces is accompanied by a USM consistent with the FACE SDM and defines its interfaces in terms of the FACE Data Model Language. In other words, for each UoC is needed a representation of the data exchanged with the other UoCs, in terms of type of data, channels of

communication and possible encryption of the information. A Domain-Specific Data Model (DSDM) captures content relevant to a domain of interest and can be used as a basis for USMs.

2.7.1 Data Model Language

In order to achieve FACE conformance, a Data Model [15, p. 4] is provided to represent data in different levels of abstraction and the relations between the different types of data. The Data Reference Architecture does not specifically follow many of the more traditional approaches in use today although it builds upon the concepts of message modelling and software modelling. The FACE Data Architecture is not traditional message based data model: the goal is to capture semantic and syntactic information by introducing different levels of abstraction in order to represent different aspects. The separation in levels provides a separation of concerns fostering reuse of the data models and supporting interoperability requirements. As the level of abstraction decreases in order to go down the hierarchy, also the ability for the model to be reused suffers the same behaviour. *NOTE*: The concepts addressed in this section bear certain similarities to those found in the field of Database Management. A reader with even a basic understanding of that domain should encounter no significant difficulties in comprehending the subsequent paragraphs.

2.7.2 Data Model Levels

The first level is the Conceptual Level which constitutes the CDM (Conceptual Data Model): the basis element at this layer are the Conceptual Entity and the Conceptual Association which can be established between Entities or between an Entity and another Association. The second level is the Logical Level which constitutes the LDM (Logical Data Model): the basis element at this layer are the Logical Entity and the Logical Association which respectively refine the Conceptual counterpart. The third level is the Platform Level which constitutes the PDM (Platform Data Model): the basis element at this layer are the Platform Entity and the Platform Association which respectively refine the Logical counterpart adding information about data representation at code level. To make an analogy with the database fields we mentioned before: Entity is the Table which represent some real world concept, both an object or information of any value as it exists, Association is the Relationship between Entities as described in the *Entity-Relationship* paradigm, while the database Query is, in the Share Data Model, represented by the View, which is a representation of the information we need.

2.7.2.1 Conceptual Data Model

The Conceptual Data Model (CDM) is the highest level of abstraction of the representation for Data involved and exchanged between the UoCs; this levels captures the semantic meaning of data elements defining *Observable* which are used to describe

Characteristics (which reminds of the concept of Attribute in Object-Oriented Programming) for each *Entity* (which reminds of the concept of Class in Object-Oriented Programming). Entities can be related via *Associations* being defined as *Participants*.

2.7.2.2 Logical Data Model

The Logical Data Model (LDM) provides a mechanism to extend each conceptual element with additional refining information such as *Units*, *Measurement* and *Coordinate Systems*. Since each Conceptual element can have multiple Logical representations, this level provides a guidelines for data sharing by means of proper conversion between *Units*, *Measurement* and *Coordinate Systems*. At this level the conversion between different reference frames is performed. As said before, the passage from Conceptual to Logical level impacts on the ability for the model to be reused.

2.7.2.3 Platform Data Model

The Platform Data Model (PDM) provides the specifics for software implementation, defining data types allowing the logical type to be represented for the specific need of the UoP. In other words, at this very level, source code for the UoP is generated using IDL for code type definition and conversion. As said before, the passage from Logical to Platform level dramatically impacts on the ability for the model to be reused.

2.7.2.4 UoP Model

The UoP models provides the means to formally specify the interfaces of a UoP in terms of data and communication characteristics; in other words, this model provides information about data passing through the interfaces of the UoP, in the sense of data exchanged coming from or going to other UoPs. The UoP model supports platform-independent specification and can be seen at both Conceptual or Logical level. The value of UoP model is that it provides description for interfaces, communications and implementation details for the integration phase.

2.7.2.5 Integration Model

The Integration model provides means to model the exchange information between UoP instances, by capturing data exchanges, view transformations and integration of UoP for documentation purpose.

2.7.3 Concluding remarks on Data Model

The FACE Data Model Language employs a multi-tiered approach to modeling entities and their relationships across conceptual, logical, and platform layers, allowing for gradual and flexible levels of abstraction. Entities, along with their attributes and associations, establish the context necessary for defining views that specify data exchanges between UoPs. The FACE Data Model Language also facilitates the

modeling of abstract UoPs, serving as a specification tool for procurement or defining elements based on the FACE Technical Standard, which can be applied in other reference architectures. To support integration between UoPs, the FACE Data Model Language includes elements for describing the overarching connectivity, routing, and transformations that are implemented within an instance of transport services. The documentation of a UoP must contain first of all:

- Programming Language(s) used in the UoP
- FACE Profile(s) in which the UoP resides
- Memory Requested by the executable

The several UoPs, a software is composed of, must communicate to each other to exchange data. Indeed, connections are needed and the related documentation must indicate:

- whether the connection is Blocking or Non-blocking
- whether the connection has a Client/Server architecture
- whether the connection has a Publisher/Subscriber architecture

Data exchanged along connections are under the form of messages and the specific structure of the messages must be documented in terms of semantic meaning of fields, unit of measurement and data types.

2.7.3.1 Motivation for FACE Data Model

Two main reasons are at the basis of creating FACE Data Model:

- Accelerate integration of software components
- Provide semantic documentation of message data, clarifying the meaning of each data

For example, through this Data Model it's very easy and immediate to check whether two systems have incompatible data in terms of representation and whether an adapter is needed or not. The Data Model provides a description of each datum in each system in order to make it as clear as possible.

Chapter 3

Proposed Solution

3.1 Introduction to the proposed solution

In order to experiment with the different layers of the FACE Standard [1], a software architecture has been proposed with the relative implementation realized and discussed in the following paragraphs in terms of software solutions and in general concepts with a proper level of detail. Thus, in the next chapter, pivotal parts of the developed code will be presented and commented in order to emphasise some architectural and algorithmic choices. The main focus during all the code design and development has been the IMA and MOSA principles of portability, reusability and interoperability which FACE embodies in its architectural guidelines, as shown in Figure 3.1. A noteworthy design could be the one proposed in the figure since the modularity of the software is emphasize by the fact that different segments could be spread over different hardware devices connected by deterministic Ethernet.

3.1.1 IMA, OSA and MOSA Principles in the proposed architecture

The architecture in Figure 3.1 demonstrates key principles of Modular Open Systems Approach (MOSA), Open Systems Architecture (OSA), and Integrated Modular Avionics (IMA):

- Modularity: The separation of applications (e.g., A/C Position app on Board 1 and EGI, GPS, and Display Apps on Board 2) and system components (such as I/O, timing services, and OS) highlights a modular structure. Each application and component can be updated or replaced independently, allowing scalability and easier upgrades.
- Interoperability: The OpenDDS middleware establishes an open communication infrastructure, promoting interoperability between modules across both boards. This use of open standards aligns with MOSA, making integration with third-party components straightforward.
- Hardware Independence: The layered structure, from applications down to the OS and middleware, abstracts the underlying hardware, embodying IMA

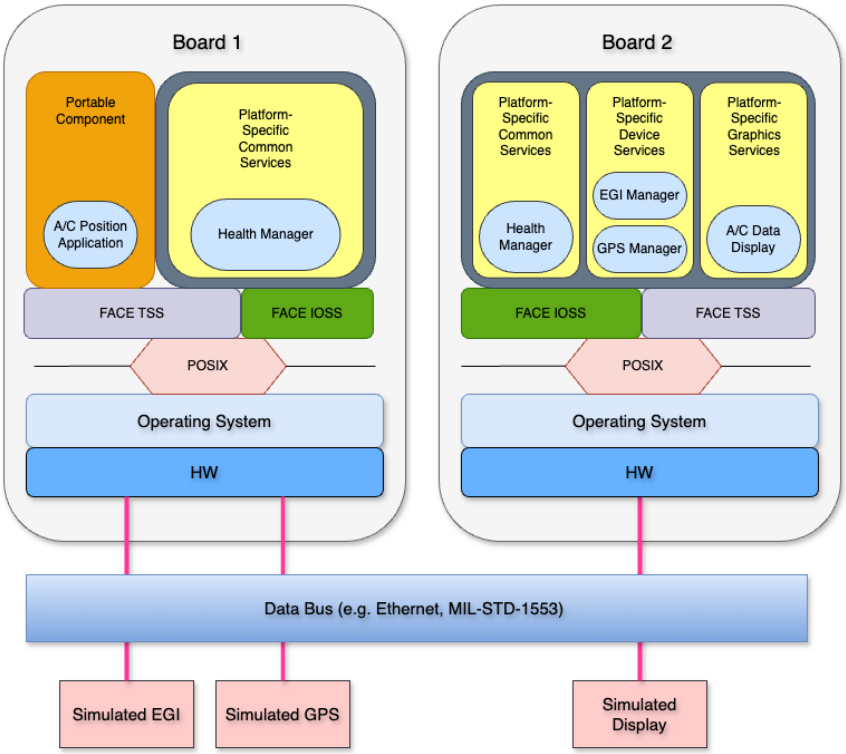

Figure 3.1: Proposal for an IMA in the context of this thesis

principles. This separation allows software modules to be transferred to different hardware platforms with minimal modification.

- Standardized Data Bus: The use of a standardized data bus (e.g., Ethernet, MIL-STD-1553) enables communication across modules from different vendors, a core MOSA and OSA concept. This design fosters reusability and vendor competition.

Purpose of this chapter

The purpose of this practical part of the thesis is to document and better understand how the different layers (see Figure 3.2), units and components work together and communicate with each other in the FACE environment. This proposed solution aims to be as much compliant with FACE Standard as possible since the obvious limitations of a limited amount of reference sources to which compare this current work.

3.2 FACE Software Architecture proposal

3.2.1 PCS

In the current software architecture the PCS contains a single UoP, which is a A/C Position app which communicates through TSS with the other segments of the stack. The applications developed for this segment must present the main feature of

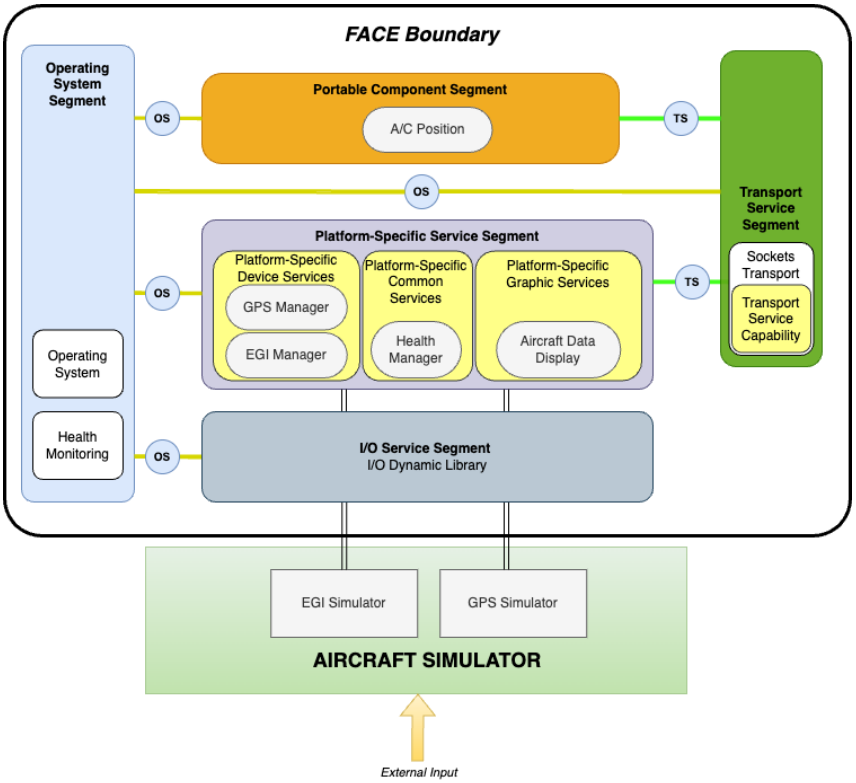

Figure 3.2: Proposal of SW architecture for this thesis

portability since they must be able to be installed on different devices and be able to work with little or no modification.

3.2.1.1 A/C Position UoP

The portability of this application is due to the fact that no specific hardware or sensor reference is made and, thus, the very same code could be installed on other devices, recompiled and used without any modification at all. To effectively achieve interoperability and portability, no bond with data transport and operating system exists, making the software agnostic to these aspects. In addition, this portability relies also on the key point that the incoming and outcoming information, via TSS, is well known and documented: the PCS applications communicating with the PSSS through TS layer are then able to understand the exchanged information and they know which topics to subscribe to or publish on to talk to other PSSS components. This PCS UoC receives the data about the aircraft from the EGI and GPS application inside PSSS via TS and performs a data fusion by means of a linear interpolation between data from EGI and data from GPS; indeed, the A/C Position (Data Fusion, DF) app makes use of the TSS interface provided my the OpenDDS implementation. The TSS abstracts the transport mechanism providing the PSSS and PCS applications several functions to exchange messages among each other. In this case, the underlying communication mechanism is realized by DDS - i.e the APIs OpenDDS provides - which realizes a publisher/subscriber model. The A/C Position app, making use of the FACE TS APIs provided by OpenDDS, opaquely instantiates two subscribers,

one from each sensor app on the other side of the TSS interface, and receives TS messages using the OpenDDS APIs. Upon receiving the data packets from both publishers, data is merged by linear interpolation and the new corrected data is thus sent to the PSGS by means of TSS. In the case of one sensor's malfunction, receiving only one packet out of two, no fusion is performed and the new data for the PSGS UoC is the very same packet just received; of course the impossibility in the data fusion is signalled to the graphic component. In the extremely rare case that both packets are missing, the last valid fused data is sent to the other application till a new acceptable info is retrieved. The communication with the PSGS application is performed via TSS on which the A/C Position application plays the role of the publisher while the PSGS is the subscriber of the related topic.

3.2.2 TSS

As anticipated in paragraph 2.5.4, this segment makes large use of the TS interface provided by OpenDDS. These APIs, declared in the header FACE/TS.hpp inside the OpenDDS folder, are FACE compliant and present the same name and signature of the functions described in the Technical Standard [1, pp. 291-300]. The FACE realization of TSS provided by OpenDDS, by means of a set of APIs, is slightly different from the DCPS implementation: the former indeed makes use of all the services the latter offers, but it hides all the low level mechanism such as creating participants and topics, type specific variable and so on. In addition, DCPS is still an implementation of the DDS middleware which focuses on data centrality while the FACE realization focuses on providing the service as FACE Technical Standard prescribes. In other words, DCPS and FACE realization are two different approaches to implement DDS technique.

3.2.2.1 Configuration File

Lots of these details, which in DCPS implementation must be hand written in the code, in the FACE TS implementation provided by OpenDDS are enclosed into a human-readable `.ini` configuration file in which topics, network protocols, and QoS policies are hard-coded. This is supposed to be a key point in the use of OpenDDS providing FACE TS implementation since the configuration is changed by means of editing the configuration file instead of having to act on the code itself recompiling it. Indeed, all the information about the DDS connection are collected in a single file and not spread in the code. The key words in the `.ini` file, furthermore, are the same used in the DCPS configuration functions: the commodity of using the `.ini` file is that all parameters are automatically set when connection is initialized, while, in standard DCPS, a pretty large amount of type specific variable needs to be declared, initialized and their methods used with precise values.

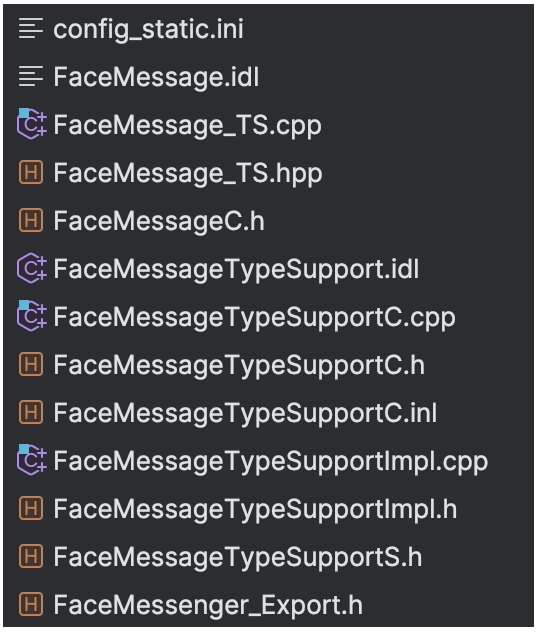

Figure 3.3: Message type declaration support files

3.2.2.2 IDL

A publisher-subscriber application from OpenDDS is realized by means of several files. Since TSS is transport layer, the most important one is of course the IDL (Interface Definition Language, `.idl`) which all the topics and data structures exchanged during the communication are defined in high level language. By means of the MPC (Make Project Creator) provided by OpenDDS, IDL data (topics and data structures such as `struct` or `union`) are automatically converted into C++ files and headers in accordance with the IDL-Language mapping. A topic in a IDL file is a data structure such as union or struct identified by the keyword `@topic`. Topics might have a key to identify the instance of the topic in a conversation; in this case the topic fields are preceded by the tag `@key` [8, Documentation/Getting Started]. Once a TSS project is compiled, starting from IDL new defined types, new header and source files are automatically created providing C level data type definition and functions declaration. These files are to be included and linked in a TSS project in order to use the new data type. In the next chapter, the IDL file produced for the software implementation of this thesis is provided and commented.

3.2.2.3 TSS support files

As mentioned before, the compilation of IDL brings to the creation of several new source and header files to realize and support new data type. The set of the produced support files is displayed in Figure 3.3.

3.2.3 PSSS

In this layer, we find the EGI and GPS apps as part of the Platform Specific Device Services (PSDS). The Platform Specific Graphics Services (PSGS) are represented

here by the graphical component acting as a display to show aircraft information and the Platform Specific Common Services (PSCS) consists in the System Level Health Monitoring, here called Health Manager, which constantly checks whether all application are correctly running or an application failure has occurred.

3.2.3.1 EGI & GPS

In the current solution, here proposed, two sensors are simulated in order to collect info provided by a very basic flight simulator, process it and send it to the PCS A/C Position UoP via TSS. These two sensors information is managed by two different applications, one for each sensor, inside the PSSS, which receive data carrying information about speed, velocity and position of the plane. The communication with the simulated sensors is made upon serial device and the reading from the serial port is carried out every 15 ms in order to simulate a sampling by the application over the serial bus.

3.2.3.2 DPM

The Device Protocol Mediation resides inside the Platform-Specific Common Services (PSCS) subsegment and allows UoCs for exchanging data between entities which use different communication protocol [1, pp. 67-69]. DPM UoC communicates with other PSSS UoCs by means of I/O lib, alias IOSS, while uses the OSS for communication outside the PSSS layer. In the proposed implementation of this thesis, a DPM UoC is realized with the only goal of retrieving data by means of FTP from a server and pass them to the EGI or GPS modules inside the PSSS. The operation for data retrieval from a server is made possible using POSIX sockets (thus OSS assisted) while the communication between the UoCs of PSSS and the DPM UoC is realized with IOSS serial management. In case of failure for the serial connections between the simulation modules and the PSSS EGI and GPS applications, the DPM intervenes in order to supply the missing data to the PSSS UoCs reading them from a backup storage in which the information has been previously saved by the simulated sensors. In other words, when an error in the serial connection is detected and packets are not delivered, after a certain amount of time, the PSSS application interested by the accident can query a storage to retrieve the info and send them to the next step in the data pipeline. In the current version, a minimal implementation of this component has been realized and, since it is in a prototypal stage, by means of conditional compiling, it has been deactivated. Thus, it doesn't appear in the PSSS realized applications. Obviously, in a future version, a fully functioning implementation of this service would be very fundamental to achieve a greater software robustness.

3.2.3.3 PSGS Display

On the main graphical widget, all the data treated by the A/C Position app in PCS are showed in real time. The ultimate purpose of this application is to show on the screen the result of the data fusion algorithm and to validate the correct functioning

of all the components involved in the data pipeline starting from the sensors simulator modules up the PCS UoP. This main window is divided vertically into two panels: the one on the left represents the aircraft on a 2D map providing also the info about latitude, longitude, altitude, speed, attitude and sensors status; the one on the right provides info about pitch, roll and yaw by means of an artificial horizon. The data to be displayed is retrieved by TSS: the PSGS component is a subscriber of a precise topic while, on the other end, the A/C Position app publishes merged data on the same topic. In the 2D map it's possible to show the position retrieved by the merged data and the positions provided by the single sensors to check the validity of the data fusion operation. In addition several labels allow to continuously keep under check the status of all the applications (both sensor modules and PSSS UoCs) and being notified if something wrong happens.

3.2.3.4 System Level Health Monitoring

Inside the Platform-Specific Common Service (PSCS) subsegment, the Health Monitoring system finds place: it is the UoC whose is charge of checking the correct behaviour of every application and, in case of malfunctioning, it must kill the faulty process and launch it again. This component, at the very beginning, launches the other software components of the software architecture and registers the *pids* in order to be able to send a killing signal to a process when it stops working properly. Communication with the other PSSS components is made with Serial while with PCS components is through TSS, via dedicated topics.

3.2.4 IOSS

As anticipated before in this chapter, in the current software implementation, several serial connections are established: one is between each sensors simulator and the PSSS apps to manage them, and one is between each PSSS UoP and the PSSS Health Manager to exchange status information.

3.2.4.1 Serial

Since communications within the PSSS and PCS segment are expected to be via IO lib, the easiest way to implement such communications would have been through serial bus since is one of the simplest protocol. On the personal computer environment, on which all the software for now runs, serial ports are represented by particular file descriptors and can be read and written as normal files by means of standard POSIX APIs. Thank to the Operating System, in fact, all the hardware details of the serial connection are hidden by means of system drivers which translate from electrical level to byte stream level. This allows the code to be compatible to every type of serial device - such as RS232 or RS422 - provided that the corresponding driver is installed in the machine the software architecture runs on. Indeed, the Serial class implemented in this context, is nothing but a wrapper of POSIX system calls with the goal of realizing the FACE IOSS functions declared in the Technical Standard

[1, pp. 247-280]. In order to simulate serial communication inside the operating system, without reading from a actual serial bus, which from the personal computer perspective makes no difference, the CLI (Command Line Interface) or terminal is needed. All connections are virtualized by the terminal and data flow is internal to the software; no edit would be need in a deployment for a real scenario since the serial operation would be managed by the operating system. In the current case, however, since the name of the serial port are assigned once the port is created, no possible hardcoding of the serial port name can be done, thus, an additional step is required. Once the ports are created (see 4.2.4.1), the assigned names must be written inside the configuration files for each serial endpoint. In case of serial hardware involved, serial port name could be written inside the code since known a priori.

NOTE: only the main methods for the Serial have been implemented and lots of unnecessary functions have been neglected or have been given a minimal implementation (which remains unused for the purpose of this thesis code). A functionalities extension and a broader support must be considered for future work.

3.2.5 OSS

The OSS segment contains the Operating System along with Language Run-Time, Application Framework, Health Monitoring Fault Management (HMFM) and Device Drivers. In the current work, a minimal version of OSS is implemented since a complete one, with all the services aforementioned, would require particular RTOS provided by licensed software, such as DEOS DDC-I RTOS [16] or LYNX OS [17]. Indeed the current work is developed on Linux OS using POSIX APIs and not a RTOS: this implies a limited use of OS services to POSIX APIs and a complete lack of task separation paradigm. However, the extensive use of POSIX APIs makes the realized software architecture portable on different RTOS platforms which are POSIX compatible by default of by means of some software extensions. A prototype version using FreeRTOS is among the future works intended to reach a more realistic deployment scenario for the software here discussed.

3.2.6 FACE Boundary External Component

3.2.6.1 Flight Simulator

The goal of this application is providing pseudo realistic data to FACE data pipeline across the architecture segments, starting from the input provided by acting on the command window, i.e. converting throttle and yoke values into speed, acceleration and position values. By adjusting positional (latitude, longitude) and attitude data (pitch, roll), along with controls like throttle and yoke inclination, a full simple dynamic simulation is achieved. Yoke movements control pitch and roll angles, enabling directional changes and altitude adjustments. Though simplified, the system provides dynamic data resembling flight behaviour, with roll affecting heading based on an empirical formula that includes speed, rather than a strict adherence to real

flight mechanics. The focus here is only on providing dynamical data to be passed towards the FACE data pipeline to check the correct functioning of each segment.

3.2.6.2 EGI and GPS simulators

The yoke and throttle inputs are continuously read by a data simulator becoming the initial values for the calculation of dynamical data to provide to EGI simulator and to GPS simulator. In this scenario, where several simplifications are made, it has been assumed that GPS receives latitude, longitude, altitude, speed, and heading directly, with added noise, while the EGI only receives speed, heading, and attitude, requiring additional calculations for position, simulating the inertial behaviour of the sensor strongly dependant from inertial data and initial conditions. Since, both modules use similar logic and same data input, their data align closely, but noise is introduced to have different simulated telemetry by the two sensors. In order to have only one reliable measurement, EGI and GPS values, after having been collected by the PSSS EGI and GPS UoCs, are sent over TSS to the A/C Position application in PCS to be filtered and merged together.

3.2.6.3 Backup

The simulated aircraft data, beside being sent to the A/C Position UoP, are stored into a backup system allowing the PSSS sensors UoCs to recover information in case of failure for the serial connection; this permits the EGI or GPS manager to be, in any case, able to provide data to A/C Position. This is managed by a software component that saves a backup of coordinates every second to an FTP server. Each time new coordinates are computed, they're serialized into a structured record with a timestamp and saved as a text file, which is then uploaded to the server. The file is overwritten with each update, enabling the DPM to retrieve only the latest data and deliver them to the EGI or GPS in case of connection issues.

3.2.7 Partitioned Architecture

In order to guarantee greater modularity and separation, a partitioned architecture can be designed by using ARINC 653 architectural standard. The aim of this standard is, indeed, to increase safety, security and reliability in critical systems such the avionics ones. ARINC 653 presents the following advantages:

- Partition Isolation: Ensures applications in separate partitions can't interfere with each other, enhancing safety and data protection.
- Reliability and Safety: Isolated architecture prevents errors in one partition from impacting others, vital for avionics systems' safety.
- Simplified Certification: Facilitates compliance with standards like DO-178C, allowing separate certification of individual modules.

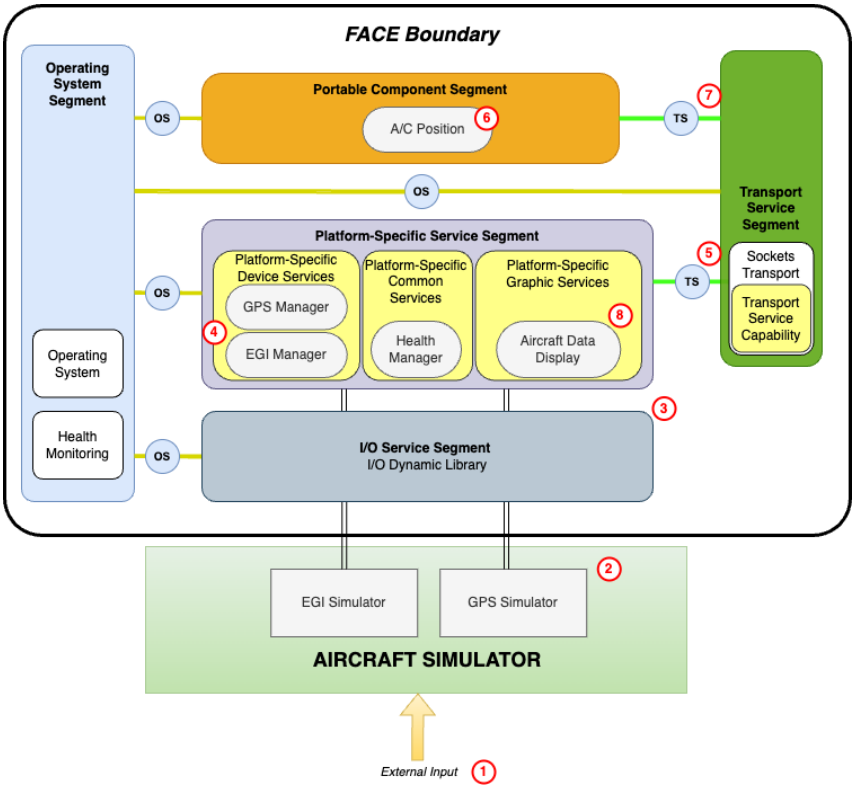

Figure 3.4: Data Path along the software architecture

- Efficient Resource Management: Allocates CPU and memory per partition, optimizing resource use and ensuring each application has what it needs.
- Modularity and Reusability: Enables software modules to be reused across systems, reducing costs and simplifying development.
- Real-Time Support: Provides strict timing control, ensuring real-time applications meet their deadlines effectively.

Unfortunately, because of the lack of an ARINC 653 platform, all these topics could not have been analyzed in the proper manner. Since this aspect is of paramount importance for a robust and efficient FACE software architecture implementation, the realization of a partitioned ARINC 653 environment is of course one of the priority goal for future work. However, a proposed architectural design could be realized in order to visualize how different software components would be hosted on different partitions and even on different boards and how they would communicate. For such a design proposal see Figure 3.5.

3.3 Proposed Data Model

Figures 3.6, 3.7 and 3.8 are vertical representations of *attitude*, *speed* and *position* Observables across the Conceptual, Logical and Platform Levels. The passage from a level of abstraction to the immediately lower one is made through realization which adds information to the observable and realizes it in the meaning of *"makes it more*

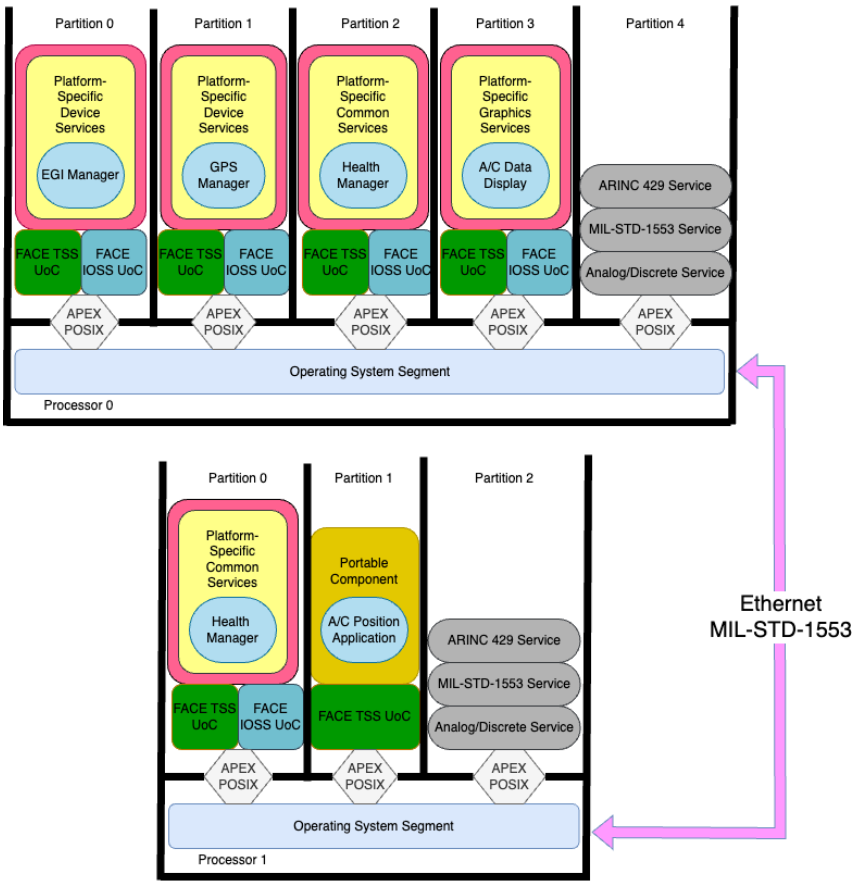

Figure 3.5: Design of a IMA architecture installed onto two communicating boards

concrete and more understandable defining additional properties". In Figure 3.6 the concept of position as latitude, longitude and altitude is realized starting from three different Observables: *Position*, *Angle* and *Distance*; the first one is thought as the main concept while the others are physical quantities involved in the main Observable. When it comes to Logical level, an Observable is realized as the measurement of that Observable and as the measurement axis for that Observable. This is what happens to *Position* which is turned into *PositionMeas* and into the three axis along which the position is measured. In turn, each axis is realized as a Platform type, a data type fit to contain the specific datum. In the end, also the *PositionMeas* is turned into its Platform counterpart becoming a structure with a list of fields representing the axis of the measurement itself in the programming language the data is coded. The very same description can be made for Figure 3.7 in which two Conceptual Observables are realized, *Attitude* and *Angle*, where the former is the main concept to be represented while the latter is the physical quantity composing its own fields.

A further explanation is needed for figure 3.8 since the Observable of *Speed* is first specialized (through the action of Specialization) into the Observables *Speed_AC* and *Speed_NED* and then realized as usual. This is similar to the concept of abstract classes in object-oriented programming in which a class represents a thing but other classes extend the parent class in order to add different information to each children keeping a common root.

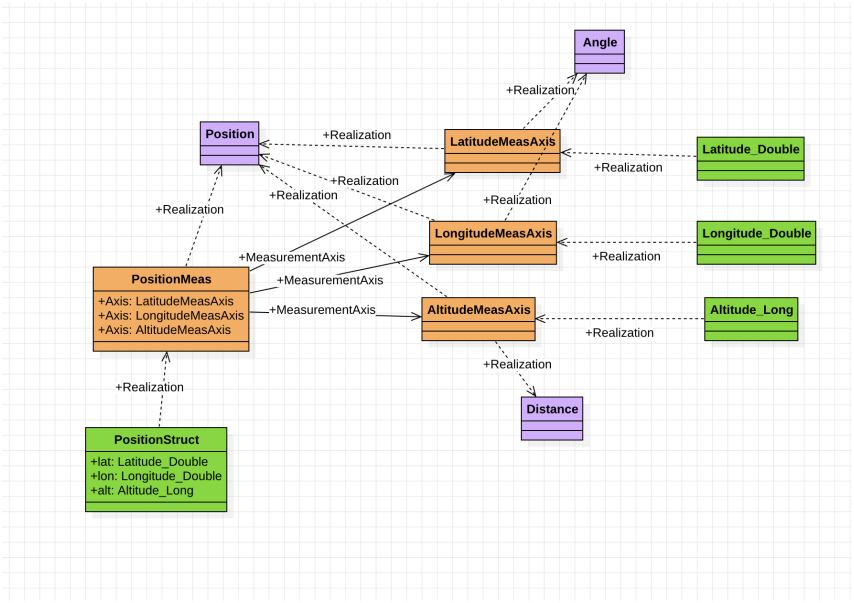

Figure 3.6: SDM for Position Entity

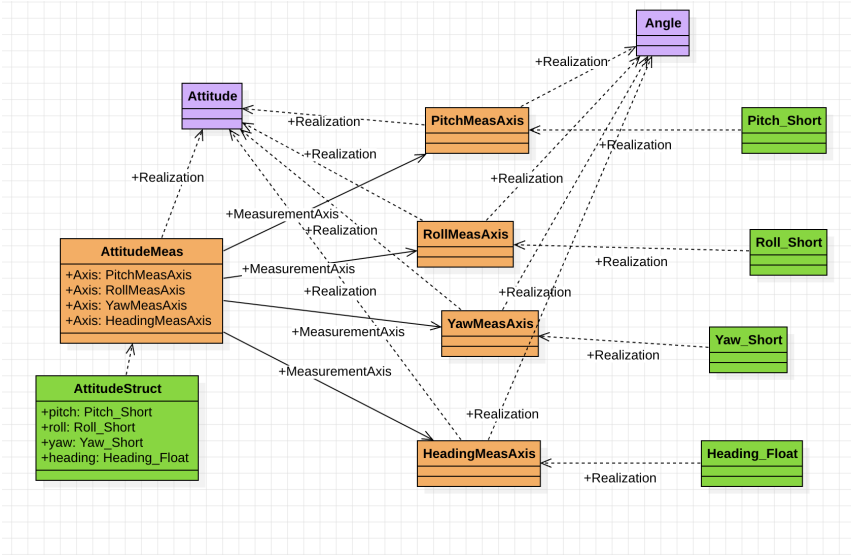

Figure 3.7: SDM for Attitude Entity

3.4 FACE OSS Profiles and Limitations

Since the main purpose has been to get a re-configurable software component able to be installed into a different board in the case of failure of the first one, it is necessary to be able to restart the application reallocating it on a second board. This means creation of a new process and, to do so, a set of POSIX APIs are needed: according to this set we can evaluate into which profile this operation falls. First of all, in order to run a new instance of the application, a new process must be launched by means of *fork*() and *exec*(). Now that we've created the new process and installed the image of the application, we may need to launch some threads in order to exploit parallelism and divide tasks: to do that a call to *pthread_create*() must be performed.

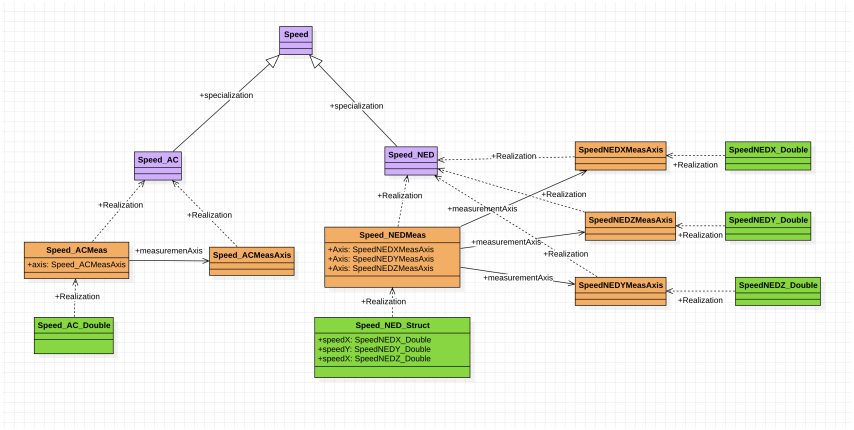

Figure 3.8: SDM for Speed Entity

For an example of implementation of process restarting after failure, go check the one provided in the paragraph 4.2.3.4 for PSSS Health Monitoring.

As just explained, many POSIX APIs have been used to develop the code using the resources and the services offered by the operating system. These different system call belong to different FACE OSS Profiles and to establish the profile into which the software falls it's necessary to check carefully all the APIs and decide accordingly to the most restrictive one. Thanks to code simplicity and reduced depth of software structure, no General Purpose exclusive APIs have been used. Almost all the used POSIX system calls are in the Security profiles but the ones used especially in the HM in PSSS, such as *kill* and *fork* cause the software to be in the Safety Extended profile.

3.5 ICD for Exchanged Data

In the following subsections, the GPS and EGI ICDs are presented in order to clarify which data are exchanged between the simulator and the PSSS through IOSS and which type of data representation is used for each piece of information. An *Interface Control Document* (ICD) defines the data format, protocols, and specifications for communication between system components, ensuring clear, consistent interactions across interfaces. Several attempts have been performed in order to understand which configuration of bits length was a good trade off between precision and efficiency in the transmission.

3.5.1 GPS ICD

The following table outlines the Interface Control Document (ICD) for GPS data acquisition, detailing key variables such as name, units, range, bit length and any applied Scale Factor. This ICD ensures clarity and consistency in integrating GPS data into the system. *NOTE*: GPS packet is 116 bits long.

Parameter	**Units**	**Range**	**Type**	**Bits Len**	**Bits Pos**	**Notes**
Latitude	semi-circles	$-1 \div 1 - 2^{-31}$	2's complement	32	$0 \div 31$	MSB = -2^0 LSB = 2^{-31} positive = North
Longitude	semi-circles	$-1. \div 1 - 2^{-31}$	2's complement	32	$32 \div 63$	MSB = -2^0 LSB = 2^{-31} positive = East
Altitude	feet	$-65536 \div 65536 - 2^{-3}$	2's complement	20	$64 \div 83$	MSB = 2^{16} LSB = 2^{-3} positive = above MSL
Speed	m/s	$0 \div 1000$	unsigned binary	16	$84 \div 99$	MSB = 2^{15} LSB = 2^0
Heading	semi-circles	$-1 \div 1 - 2^{-15}$	2's complement	16	$100 \div 115$	SF=180 MSB = -2^0 LSB = 2^{-15} positive = North

Table 3.1: GPS ICD

3.5.2 EGI ICD

The following table outlines the Interface Control Document (ICD) for EGI data acquisition, detailing key variables such as name, units, range, bit length and any applied Scale Factor. This ICD ensures clarity and consistency in integrating EGI data into the system. *NOTE*: the EGI packet is 308 bits long.

Parameter	**Units**	**Range**	**Type**	**Bits Len**	**Bits Pos**	**Notes**
Latitude	semi-circles	$-1 \div 1 - 2^{-31}$	2's complement	32	$0 \div 31$	MSB = -2^0 LSB = 2^{-31} positive = North
Longitude	semi-circles	$-1 \div 1 - 2^{-31}$	2's complement	32	$32 \div 63$	MSB = -2^0 LSB = 2^{-31} positive = East
Altitude	feet	$-65536 \div 65536 - 2^{-3}$	2's complement	20	$64 \div 83$	MSB = 2^{16} LSB = 2^{-3} positive = above MSL
Speed	m/s	$0 \div 1000$	unsigned binary	16	$84 \div 99$	MSB = 2^{15} LSB = 2^0
Heading	semi-circles	$-1 \div 1 - 2^{-15}$	2's complement	16	$100 \div 115$	SF=180 MSB = -2^0 LSB = 2^{-15} positive = North
Pitch	semi-circles	$-1 \div 1 - 2^{-15}$	2's complement	16	$116 \div 131$	SF=30 MSB = -2^0 LSB = 2^{-15}
Roll	semi-circles	$-1 \div 1 - 2^{-15}$	2's complement	16	$132 \div 147$	SF=30 MSB = -2^0 LSB = 2^{-15}
Yaw	semi-circles	$-1 \div 1 - 2^{-15}$	2's complement	16	$148 \div 163$	SF=180 MSB = -2^0 LSB = 2^{-15}

Parameter	**Units**	**Range**	**Type**	**Bits Len**	**Bits Pos**	**Notes**
Vel_x	feet/s	$-8192 \div 8192 - 2^{-18}$	2's complement	32	$164 \div 195$	MSB = -2^{13} LSB = 2^{-18} positive = North
Vel_y	feet/s	$-8192 \div 8192 - 2^{-18}$	2's complement	32	$196 \div 227$	MSB = -2^{13} LSB = 2^{-18} positive = East
Vel_z	feet/s	$-8192 \div 8192 - 2^{-18}$	2's complement	32	$228 \div 259$	MSB = -2^{13} LSB = 2^{-18} positive = Up
Acc_x	feet/s^2	$-1024 \div 1024 - 2^{-5}$	2's complement	16	$260 \div 275$	MSB = -2^{10} LSB = 2^{-5}
Acc_y	feet/s^2	$-1024 \div 1024 - 2^{-5}$	2's complement	16	$276 \div 291$	MSB = -2^{10} LSB = 2^{-5}
Acc_z	feet/s^2	$-1024 \div 1024 - 2^{-5}$	2's complement	16	$292 \div 307$	MSB = -2^{10} LSB = 2^{-5}

Table 3.2: EGI ICD

Chapter 4

Implementation Details

4.1 Introduction

As anticipated in the previous chapter, in order to achieve a deep knowledge of the advantages brought in by the FACE Technical Standard from the software point of view, a fully functional software architecture implementation has been realized. A part of the developed code will be presented and commented in details but, for brevity reasons, only the most significant excerpts will be shown casting light onto the fundamental concepts. The purpose of this implementation part of the thesis is to document and better understand how the different layers, units and components work together and communicate with each other within the FACE environment. The realized code, given its own modular nature, can be deployed on different CPUs (as shown in Figure 4.1) where the different FACE segments are spread on different boards linked by means of a bus on which a serial communication is used to represent the information sent using the IOSS and IO Services provided. Within the same board, instead, different UoCs communicate with each other by means of TSS or IOSS connections according to the FACE Standard.
NOTE: the software structure of the proposed architecture is reported in Appendix B where all the produced files are listed divided by the software component they realize.

4.2 FACE Layers

Before starting with the analysis of the FACE segments and the implemented solution which realizes the algorithms presented in the previous chapter, additional information is needed. Both the proposed solution and its implementation have been realized for two operating systems since not all the external provided APIs - such as those of OpenDDS - were compatible with others inside a segment rather that another one. Thus, according to operating systems, software component have been placed in different segments. The original software solution was developed for MacOS and contained, inside the PCS, the graphical component to display the A/C information. The second version, instead, hosts the graphical widget inside the PSGS (the Platform-

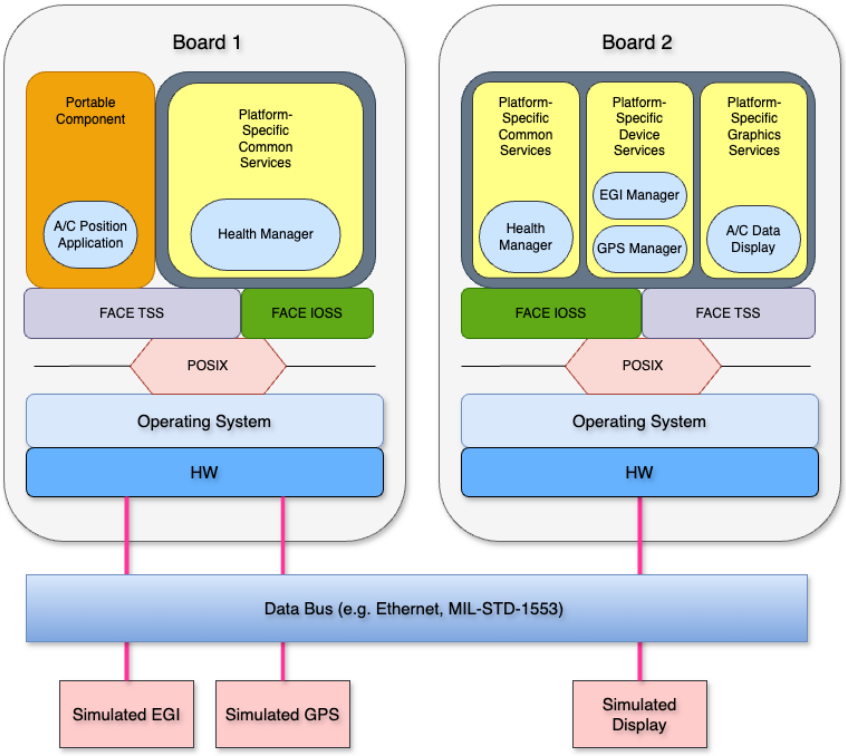

Figure 4.1: IMA Architecture proposal

Specific Graphics Services) subsegment. This porting from one operating system to another will be discussed in more details in the next chapter, but an introduction to this aspect here was necessary.

4.2.1 PCS Implementation

The PCS hosts the Aircraft (A/C) Position Application which performs the computation of the plane position by means of data fusion filtering and merging data from PSSS UoCs.

4.2.1.1 A/C Position UoP

```
void signal_handler(int signal, siginfo_t *info, void *extra){
    if(signal==SIGKILL){
        /*
        close Serial connections and destroy TS connections
        */
    }
}

int main() {

    // signal handler instantiation to manage signals
    sa.sa_sigaction = signal_handler;
    ...

    // initialize TS service
    FACE::RETURN_CODE_TYPE status;
    FACE::TS::Initialize(...);


    // create TS connections with EGI GPS and HM
    FACE::TS::Create_Connection(
```

```
            "sub_EGI", FACE::PUB_SUB, connId_egi, dir_egi,
            max_msg_size_egi, FACE::INF_TIME_VALUE, status);

    FACE::TS::Create_Connection(
            "sub_GPS", FACE::PUB_SUB, connId_gps, dir_gps,
            max_msg_size_gps, FACE::INF_TIME_VALUE, status);

    FACE::TS::Create_Connection(
            "pub_Monitor", FACE::PUB_SUB, connId_mnt, dir_mnt,
            max_msg_size_mnt, FACE::INF_TIME_VALUE, status);

    auto start = std::chrono::high_resolution_clock::now();

    char buff[30];
    while(1){

        ...

        // receive messages
        FACE::TS::Receive_Message(
                connId_egi, 1.5e8, txn_egi, msg_TS_egi,
                max_msg_size_egi, status_egi);


        FACE::TS::Receive_Message(
                connId_gps, 1.5e8, txn_gps, msg_TS_gps,
                max_msg_size_gps, status_gps);



        // check whether EGI and GPS messages are ok
        if (status_egi == FACE::NO_ERROR && status_gps == FACE::NO_ERROR) {

            if(msg_TS_egi.timestamp>0 && msg_TS_gps.timestamp>0) { //BOTH

                // both packet were recieved => fusion
                msg.fusion = FACE::BOTH;
                msg.tmp = time(nullptr);

                msg.ac_speed = (msg_TS_egi.speed + msg_TS_gps.speed) / 2.0;

                msg.att.pitch = msg_TS_egi.att.pitch;
                msg.att.roll = msg_TS_egi.att.roll;
                msg.att.yaw = msg_TS_egi.att.yaw;
                msg.att.hd = (msg_TS_egi.att.heading + msg_TS_gps.att.heading)
     / 2.0;

                msg.pos.lat = (msg_TS_egi.pos.lat + msg_TS_gps.pos.lat) / 2.0;
                msg.pos.lon = (msg_TS_egi.pos.lon + msg_TS_gps.pos.lon) / 2.0;
                msg.pos.alt = (msg_TS_egi.pos.alt + msg_TS_gps.pos.alt) / 2.0;

                msg.speed.X = msg_TS_egi.speed_NED.X;
                msg.speed.Y = msg_TS_egi.speed_NED.Y;
                msg.speed.Z = msg_TS_egi.speed_NED.Z;

                msg.acc.X = msg_TS_egi.acc_NED.X;
                msg.acc.Y = msg_TS_egi.acc_NED.Y;
                msg.acc.Z = msg_TS_egi.acc_NED.Z;

                /*
                 * EGI
                 */

                msg.egi = msg_TS_egi;
                ...

                /*
                 * GPS
                 */

                msg.gps = msg_TS_gps;
                ...


            }
```

```
96              else if(msg_TS_egi.timestamp>0 && msg_TS_gps.timestamp==0){  //EGI
97
98                  // no fusion; EGI only data copied into packet
99                  msg.fusion = FACE::EGI;
100                 msg.tmp = time(nullptr);
101
102                 msg = msg_TS_egi;
103
104                 ...
105
106
107                 //EGI ONLY
108                 msg.egi = msg_TS_egi;
109                 ...
110
111
112             }
113             else if(msg_TS_egi.timestamp==0 && msg_TS_gps.timestamp>0){ //GPS
114
115                 // no fusion; GPS only data copied into packet
116                 msg.fusion = FACE::GPS;
117                 msg.tmp = time(nullptr);
118
119                 msg = msg_TS_gps;
120                 ...
121
122                 //GPS ONLY
123                 msg.gps = msg_TS_gps;
124                 ...
125
126
127             }
128             else{
129                 // send the same old message
130                 msg.fusion = FACE::NONE;
131                 ...
132             }
133         }
134
135         /*
136          * CASE EGI
137          */
138         else if (status_egi == FACE::NO_ERROR && status_gps != FACE::NO_ERROR)
    {
139             perror("EGI ONLY -- GPS reception error!");
140         }
141
142         /*
143          * CASE GPS
144          */
145         else if (status_egi != FACE::NO_ERROR && status_gps == FACE::NO_ERROR)
    {
146             perror("GPS ONLY -- EGI reception error!");
147         }
148
149         /*
150          * NO reception at all: both timeouts expired
151          */
152         else{
153             perror("No reception at all!");
154         }
155
156
157         auto end = std::chrono::high_resolution_clock::now();
158         auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(
    end - start).count();
159
160         // send each second a message to HM
161         if(duration>1e3){
162
163             ...
164
165             FACE::TS::Send_Message(
166                     connId_mnt, FACE::INF_TIME_VALUE, txn_mnt, msg_TS_monitor,
167                     max_msg_size_mnt, status_monitor);
```

```
168
169
170             start = std::chrono::high_resolution_clock::now();
171         }
172
173
174     }
175
176     close_connection(socket_handle, status);
177     ...
178
179
180     return 0;
181 }
```

This PCS UoP receives data from the EGI and GPS of the PSSS via TS and performs position calculation, by means of data fusion, using an arithmetic mean of the two sources. The A/C Position app receives messages from both sensors UoCs using TSS without any knowledge of the underlying DDS mechanism. When data packets from both sensors are available, they are merged and the resulting corrected data is sent to the graphical widget in the PSGS via TSS. If one sensor fails and only one packet is received, no fusion is performed, and the single available packet is published onto the topic from which the PSGS component reads to display A/C data; in this case, an alert indicating the fusion issue is showed on the display. If both packets are missing, the last valid fused data is published until new data is available. Communication with the graphical component residing inside the PSGS is made through the TSS: the A/C Position application publishes the `Position_DF` topic and the PSGS UoC, in turn, subscribes to it.

4.2.2 TSS Implementation

As anticipated in paragraph 2.4.1, this segment makes large use of the FACE TSS implementation provided by OpenDDS. These APIs, declared in the header FACE/TS.hpp inside the OpenDDS folder [8], are FACE compliant and present the same name and signature for the provided functions. To enable the use of these provided APIs, it is necessary to include both headers and the manually compiled dynamic libraries into each project, which aims to use TSS.

4.2.2.1 FACE/TS.hpp

It is now reported the content of the FACE/TS.hpp file in order to give a look at the main TSS functions as declared inside the FACE Technical Standard [1].

```
1  #ifndef OPENDDS_FACE_TS_HPP
2  #define OPENDDS_FACE_TS_HPP
3  #include "common.hpp"
4  #include "TS_common.hpp"
5  #include "OpenDDS_FACE_Export.h"
6
7  namespace FACE {
8  namespace TS {
9
10   void Initialize(
11     /* in */ const CONFIGURATION_RESOURCE configuration_file,
12     /* out */ RETURN_CODE_TYPE& return_code);
```

```
13
14    void Create_Connection(
15      /* in */ const CONNECTION_NAME_TYPE connection_name,
16      /* in */ MESSAGING_PATTERN_TYPE pattern,
17      /* out */ CONNECTION_ID_TYPE& connection_id,
18      /* out */ CONNECTION_DIRECTION_TYPE& connection_direction,
19      /* out */ MESSAGE_SIZE_TYPE& max_message_size,
20      /* in */ TIMEOUT_TYPE timeout,
21      /* out */ RETURN_CODE_TYPE& return_code);
22
23    void Destroy_Connection(
24      /* in */ CONNECTION_ID_TYPE connection_id,
25      /* out */ RETURN_CODE_TYPE& return_code);
26
27    ...
28  } // namespace TS
29  } // namespace FACE
30
31  #endif // FACE_TS_HPP_HEADER_FILE
```

4.2.2.2 FaceMessage_TS.hpp

Beside the FACE/TS.hpp file, which reports the main connection management functions, as described in paragraph 3.2.2.2, during the IDL compilation, several support files are generated and, among them, FaceMessage_TS.hpp finds place: this file overrides the TSS message exchange functions *Receive_Message()* and *Send_Message()* for each message type defined inside the IDL file. A brief excerpt of the file is now reported; in particular, the following example shows the aforementioned functions defined, in this case, for PSSS EGI management UoC packet exchange.

```
1  #ifndef OPENDDS_IDL_GENERATED_FACEMESSAGE_TS_HPP_3II6FW
2  #define OPENDDS_IDL_GENERATED_FACEMESSAGE_TS_HPP_3II6FW
3  #include "FaceMessageC.h"
4  #include "FACE/TS.hpp"
5  namespace FACE
6  {
7    ...
8
9    namespace TS
10   {
11     void Receive_Message(
12       /* in */ CONNECTION_ID_TYPE connection_id,
13       /* in */ TIMEOUT_TYPE timeout,
14       /* inout */ TRANSACTION_ID_TYPE& transaction_id,
15       /* out */  ::Messenger::Position_EGI& message,
16       /* in */ MESSAGE_SIZE_TYPE message_size,
17       /* out */ RETURN_CODE_TYPE& return_code);
18
19     void Send_Message(
20       /* in */ CONNECTION_ID_TYPE connection_id,
21       /* in */ TIMEOUT_TYPE timeout,
22       /* inout */ TRANSACTION_ID_TYPE& transaction_id,
23       /* inout */  ::Messenger::Position_EGI& message,
24       /* inout */ MESSAGE_SIZE_TYPE& message_size,
25       /* out */ RETURN_CODE_TYPE& return_code);
26
27     ...
28
29   }
30 }
31
32 #endif /* OPENDDS_IDL_GENERATED_FACEMESSAGE_TS_HPP_3II6FW */
```

4.2.2.3 TSS Configuration file

The OpenDDS implementation of FACE TS is slightly different from the DCPS one: the former, indeed, makes use of all the services the latter offers, but it hides all the low level mechanism such as creating participants and topics, type specific variable and so on. Lots of these details, which in DCPS implementation must be hand written in the code, in the FACE implementation are enclosed into a human-readable `.ini` configuration file in which topics, network protocols, and QoS policies are hard-coded. This is supposed to be a key point in the use of FACE TSS interface provided by OpenDDS since the configuration is changed by means of editing the configuration file instead of having to act on the code itself. Indeed, all information about the DDS connection are collected in a single file and not spread in the code. The key words in the `.ini` file, furthermore, are the same used in the DCPS configuration functions: the commodity of using the `.ini` file is that all parameters are automatically set when connection is initialized while in standard DCPS a pretty large amount of type specific variables are to be declared, initialized and their methods used with precise values. In the following, some extracts of the `.ini` file are reported, just to give an idea of the content and the structure of the this file:

```
[common]
DCPSGlobalTransportConfig=$file

# domains
[domain/3]
DiscoveryConfig=DEFAULT_STATIC

...

[rtps_discovery/uni_rtps]
SedpMulticast=0
ResendPeriod=2

# transport config
#PUB EGI
[transport/rtps_pub1]
transport_type=rtps_udp
use_multicast=0
local_address=127.0.0.1:21074

[config/pub_EGI]
transports=rtps_pub1

# SUB EGI
[transport/rtps_sub1]
transport_type=rtps_udp
use_multicast=0
local_address=127.0.0.1:21075

[config/sub_EGI]
transports=rtps_sub1

...

# topic
[topic/Position_EGI]
platform_view_guid=104
type_name=Messenger::Position_EGI
max_message_size=128

[topic/Position_GPS]
platform_view_guid=104
type_name=Messenger::Position_GPS
max_message_size=64

```

```
46 [topic/Monitor]
47 platform_view_guid=104
48 type_name=Messenger::Monitor
49 max_message_size=16
50
51 ...
52
53 [connection/pub_EGI]
54 id=1
55 participantid=111111
56 domain=3
57 direction=source
58 topic=Position_EGI
59 datawriterqos=durable_writer
60 config=pub_EGI
61
62 [connection/sub_EGI]
63 id=2
64 participantid=111112
65 domain=3
66 direction=destination
67 topic=Position_EGI
68 datareaderqos=durable_reader
69 config=sub_EGI
70
71 ...
72
73 [datawriterqos/durable_writer]
74 durability.kind=TRANSIENT_LOCAL
75 reliability.kind=RELIABLE
76
77 [datareaderqos/durable_reader]
78 durability.kind=TRANSIENT_LOCAL
79 reliability.kind=RELIABLE
```

In lines 2-12 the file contains connection configuration info. In lines 20-31, publishers and subscribers are configured to use the UDP protocol. Local address ports are specified, which will be used to establish the underlying socket in the Transport Service (TS) layer. In lines 35-49 three topics - referring to those defined in the IDL file (see paragraph 4.2.2.4) - are declared: the first and the second are ordinary topics about position info exchanged between EGI or GPS with the A/C Position application; the third topic is used by several application to communicate the status to a Health Manager application running in the PSSS. In lines 53-71, all the publishers and subscribers are defined; after that, a topic, a direction in the communication flow and a QoS policy is assigned to them. At the end, in lines 73-79, QoS for data reader and data writer are configured: the adopted options are most common in this type of communication. A durability kind of `TRANSIENT_LOCAL` means that data readers associated/connected with a data writer will be sent all of the samples in the data writer's history. The `RELIABLE` value indicates that the service should eventually deliver all values to eligible data readers. It is important to notice that if a publisher could not guarantee reliable delivery of messages, the subscriber is unable to fulfil its requirement and therefore is unable to find a match, thus no match can exist between one "reliable" participant and a "best effort" one.

The following example shows the instantiation of the subscriber embedded in the EGI application: each parameter passed to the function is given a value written inside the previously shown `.ini` file.

```
1 FACE::TS::Create_Connection(
```

```
            "sub_EGI", FACE::PUB_SUB, connId_egi, dir_egi,
            max_msg_size_egi, FACE::INF_TIME_VALUE, status);
```

By indicating `sub_EGI` in the *Create_Connection* function, the variables *connId_egi*, *dir_egi* and *max_msg_size_egi* are set with the values reported in the `.ini` file and these variables are to be used in the other APIs provided by the OpenDDS implementation of the FACE TSS to refer to the specific publisher or subscriber which sends or receives a message belonging to a particular topic.

4.2.2.4 Interface Definition Language

A publisher-subscriber application from OpenDDS is realized by means of several files. One of the most important is the `.ini` as mentioned in the previous paragraph. Another of extreme importance is the IDL file in which all the topics and data structure exchanged during the communication are presented in high level language. By means of the MPC (Make Project Creator), IDL data (topics and data structures such as `struct` or `union`) are automatically converted into C++ files and headers in accordance with the IDL-Language mapping.

In the following, the IDL file is reported:

```
module Messenger {

  struct pos_t{
    double lat;
    double lon;
    long alt;
  };

  struct att_t{
    short pitch;
    short roll;
    short yaw;
    float heading;
  };

  struct acc_t{
    double X;
    double Y;
    double Z;
  };

  struct speed_t{
    double X;
    double Y;
    double Z;
  };

  @topic
  struct Position_EGI{
    pos_t pos;
    att_t att;
    double speed;
    speed_t speed_NED;
    acc_t acc_NED;
    @key long long timestamp;
  };

  @topic
  struct Position_GPS{
    pos_t pos;
    att_t att;  /* heading only */
    double speed;
    @key long long timestamp;
```

```
44    };
45
46    @topic
47    struct Monitor{
48      long code;
49      @key long long timestamp;
50    };
51
52  };
```

Before defining the topics used in the DDS communication, some additional data types (lines 3-25) must be created as realization at Platform level of the observables depicted in Figures 3.6, 3.7 and 3.8. In the second part (lines 28-50), three topics are defined: *Position_EGI* and *Position_GPS* are used respectively by EGI and GPS apps to communicate their data to the A/C Position application via TS, while the third topic *Monitor* is used by the PCS UoP (A/C Position) to communicate its status to the Health Manager located inside the PSSS. Once compiled, this IDL file is converted into several C++ source and header files to define new data types as discussed in paragraph 3.2.2.3 and shown in Figure 3.3.

4.2.2.5 Building code that includes OpenDDS TSS Library implementation

A software application, composed of C++ source code, that includes OpenDDS interface headers, can be built using two build systems: MPC [18] or CMake, where MPC file is a specific *.mpc* file in which all the project components, i.e. executables and libraries, are listed and, for each of them, source files, header files and dependencies are indicated; indeed MPC is "The Makefile, Project, and Workspace Creator", the build system used by OpenDDS to configure the build and generate platform specific build files (Makefiles, VS solution files, etc.). In this current work, however, instead of MPC, CMakeLists.txt files are used to obtained the executables and libraries realizing the SW applications implemented throughout the development of this thesis. All the code is built in the command line terminal accessing the build folder for each software components. The commands used to generate executable or libraries are listed below:

```
1 mkdir build
2 cd build
3 cmake ..
4 cmake --build .
```

The command "`cmake ..`" configures the environment to use the CMakeLists.txt file, linking all the resources required for building an executable. The command "`cmake --build .`" then compiles the executable by gathering and linking all necessary resources. In the following, an example of `CMakeLists.txt` file used to build code embedding OpenDDS libraries is presented. In this case, the excerpt relates to the file for the build of the EGI application inside the PSSS. As explained in the very beginning of the current subsection, to enable the use of TSS APIs some libraries and

several headers must be linked into the project. The here presented `CMakeLists.txt` does so to allow the PSSS EGI management UoC to make use of the OpenDDS FACE TSS APIs to communicate with the PCS A/C Position application vis TSS.

```
cmake_minimum_required(VERSION 3.3...3.27)
project(EGI)

set(CMAKE_INCLUDE_CURRENT_DIR ON)
set(CMAKE_OSX_ARCHITECTURES "arm64")

link_directories(...)
include_directories(...)

find_package(OpenDDS REQUIRED)

opendds_target_sources(messenger_idl PUBLIC "path/to/FaceMessage.idl")
target_link_libraries(messenger_idl PUBLIC OpenDDS::Dcps)

set(OPEN_DDS_LIB_DIR path/to/OpenDDS/lib)
set(OPEN_DDS_DDS_DIR path/to/OpenDDS/dds)

link_directories(${OPEN_DDS_LIB_DIR})
link_directories(${OPEN_DDS_DDS_DIR})

set(opendds_libs
    OpenDDS::Dcps
    OpenDDS::InfoRepoDiscovery OpenDDS::Tcp
    OpenDDS::Rtps OpenDDS::Rtps_Udp
    OpenDDS::FACE
    messenger_idl
)

...

add_executable(EGI ${FACE_SERIAL_SOURCES} main.cpp EGI.cpp ${FACE_TS_SRC})
target_link_libraries(EGI ${opendds_libs} path/to/libFACE_Serial.dylib)
```

4.2.3 PSSS Implementation

In this layer, as shown in Figure 3.2, the EGI and GPS apps can be found as part of the Platform-Specific Device Services (PSDS), the DPM UoC as part of the Platform-Specific Common Services (PSCS) and the window to show the aircraft information as part of the Platform-Specific Graphics Services (PSGS). Inside the PSCS subsegment, finds place also the System Level Health Monitoring, called Health Manager, which must supervise the behaviour of the system and in case of failure report to the HMFM in the OS segment.

4.2.3.1 EGI and GPS Applications

In the current code implementation, the PSSS EGI and GPS management applications were created with the aim of interfacing with the EGI and GPS sensors via the IOSS and translating the specific sensor data, defined via an Interface Control Document (ICD), towards the FACE UoP Supplied Models (USM) or the Data model (based on the Shared Data Model) in which the data they exchange are defined. These applications are quite the same and their main goal is to receive different data from the flight simulator and send them to the PCS A/C Position app: because of modularity and strict segment separation no direct communication between the

hardware simulators and the PCS component is possible but the information must pass through the PSSS segment which is in charge of collecting data from the hardware via IO lib and send them to upper levels along the stack. Figures 4.2 and 4.3 are reporting the UML for EGI and GPS applications to visualize how the code is structured and how the different classes are interconnected.

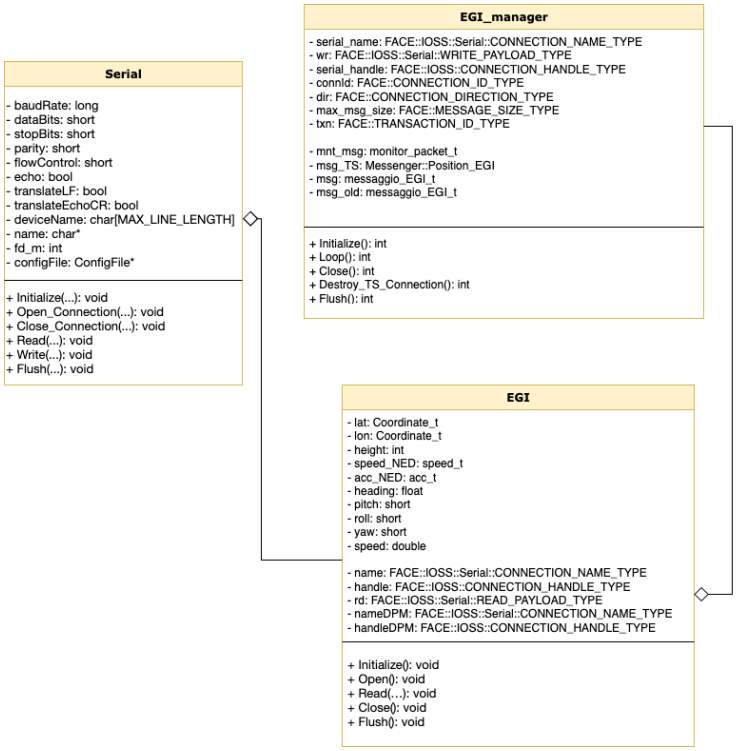

Figure 4.2: PSSS EGI application UML

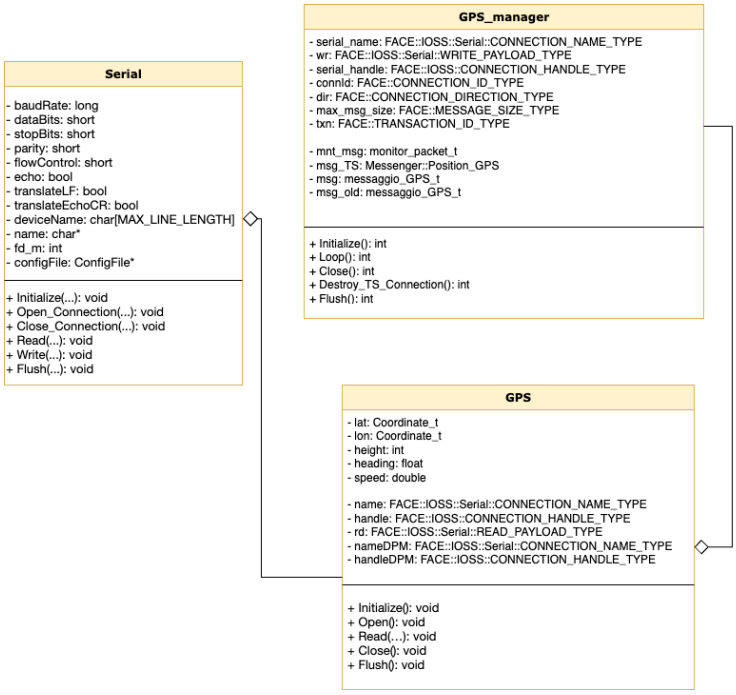

Figure 4.3: PSSS GPS application UML

4.2.3.2 EGI software in details

In the following pages, a collection of extracts of the EGI code is reported. The EGI application is composed of two classes or parts: the main makes use of an instance of the EGI_manager class, implemented as a Singleton, which communicates with the upper level via TSS and of a instance of a second C++ EGI class with the goal of communicating with the simulated EGI sensor. This last class is a wrapper containing a Serial class reference in order to read the serial bus from which to recover the produced data by the EGI simulator. This means that the EGI_manager object, in an continuous loop, uses an EGI object to read from the serial by means of IOSS APIs (provided by the IO lib) and then publishes those information to a topic the PCS A/C Position component reads from. Thus, the EGI class is a sort of serial reader while the main loop, provided by the EGI_manager perform message publishing for A/C Position application on the common topic. *NOTE*: the provided code corresponds only to the EGI PSSS application since the GPS PSSS application shares a common structure and software with due differences because of the difference in the treated information.

4.2.3.2.1 EGI Class Definition

In the following, the most important methods of the EGI class are reported: these methods have the same name of the main function of IOSS library described in subsection 4.2.4 since the former are a wrapper for the latter.

```
...
class EGI {
public:
    void Initialize();
    void Open();
    void Read();
    void Close();
private:
    // Serial object from IOSS lib to interact with the serial port
    FACE_Serial serial;
    ...
};
...
```

4.2.3.2.2 EGI::Initialize

```
void EGI::Initialize(){
    // set initial values, allocate memory for payload and retrieve config
    files
    ...
    serial.Initialize(config, return_code);
    ...
}
```

In this function all the variables are set to their initial value, the payload space for the message is allocated in the memory and the serial is initialized with the parameters contained in the `serial1_config.cfg` file. As anticipated before, the EGI methods

are wrapper for the Serial methods and thus the initialization of the EGI corresponds to the initialization of the Serial object.

4.2.3.2.3 EGI::Read

```
void EGI::Read(messaggio_EGI_t *msg, FACE::RETURN_CODE_TYPE& return_status){

    serial.Read(handle, FACE::IOSS::Serial::NO_WAIT, &rd, EGI_PACKET_SIZE,
    return_status);

    switch(return_status){

        case FACE::NO_ERROR:{

            // data extraction from byte stream
            //lat lon
            std::string str1 = string(rd.payload).substr(0,32);
            std::string str2 = string(rd.payload).substr(32,32);
            //alt
            std::string str3 = string(rd.payload).substr(64,20);
            // speed
            std::string str4 = string(rd.payload).substr(84,16);
            // head
            std::string str5 = string(rd.payload).substr(100,16);
            // pitch roll yaw
            std::string str6 = string(rd.payload).substr(116,16);
            std::string str7 = string(rd.payload).substr(132,16);
            std::string str8 = string(rd.payload).substr(148,16);
            // vel
            std::string str9 = string(rd.payload).substr(164,32);
            std::string str10 = string(rd.payload).substr(196,32);
            std::string str11 = string(rd.payload).substr(228,32);
            // acc
            std::string str12 = string(rd.payload).substr(260,16);
            std::string str13 = string(rd.payload).substr(276,16);
            std::string str14 = string(rd.payload).substr(292,16);


            // ACCORDING TO ICD
            // conversion from string to aircraft data
            double c1 = (double) ((binaryToNumeric<float>(str1, 32)) * (1 <<
    31));
            double c2 = (double) ((binaryToNumeric<float>(str2, 32)) * (1 <<
    31));

            height = (int) ((binaryToNumeric<float>(str3, 20)) * (1 << 3));

            decToDMS(c1, &lon);
            decToDMS(c2, &lat);

            speed = (unsigned int) ((binaryToNumeric<float>(str4, 16)) * (1 <<
     0));

            heading = (float) ((binaryToNumeric<float>(str5, 16)) * (1 << 15))
    ;  //in radians

            pitch = (short) ((binaryToNumeric<float>(str6, 16)) * (1 << 15));
            roll = (short) ((binaryToNumeric<float>(str7, 16)) * (1 << 15));
            yaw = (short) ((binaryToNumeric<float>(str8, 16)) * (1 << 15));

            speed_NED.X = (double) ((binaryToNumeric<float>(str9, 32)) * (1 <<
     18));
            speed_NED.Y = (double) ((binaryToNumeric<float>(str10, 32)) * (1
    << 18));
            speed_NED.Z = (double) ((binaryToNumeric<float>(str11, 32)) * (1
    << 18));

            acc_NED.X = (double) ((binaryToNumeric<float>(str12, 16)) * (1 <<
    5));
            acc_NED.Y = (double) ((binaryToNumeric<float>(str13, 16)) * (1 <<
    5));
```

```
57              acc_NED.Z = (double) ((binaryToNumeric<float>(str14, 16)) * (1 <<
    5));
58
59              msg->pos.alt = height;
60              msg->pos.lon = c1;
61              msg->pos.lat = c2;
62              msg->att.hd = heading;
63              msg->att.pitch = pitch;
64              msg->att.roll = roll;
65              msg->att.yaw = yaw;
66              msg->speed.X = speed_NED.X;
67              msg->speed.Y = speed_NED.Y;
68              msg->speed.Z = speed_NED.Z;
69
70              msg->acc.X = acc_NED.X;
71              msg->acc.Y = acc_NED.Y;
72              msg->acc.Z = acc_NED.Z;
73
74              msg->ac_speed = speed;
75
76              msg->tmp = rd.received_time;
77
78              break;
79          }
80
81          case FACE::NOT_AVAILABLE:{
82
83              //use DPM to access databse and retrieve a backup copy
84              ...
85              break;
86          }
87          ...
88      }
89
90
91  }
```

The reading operation is performed on the serial device every 15ms upon call by the main loop in order to simulate a sampling by the module on the serial connection with the hardware as the reading operation attempts to retrieve $PACKET\_SIZE$ bytes from the serial connection. According to the `return_code`, different behaviours are possible:

- FACE::NO_ERROR → the read operation has been successful and new aircraft data can be retrieved from the serial: new latitude, longitude and altitude and so on. These data are under the form of binary sequence and must be converted first into numerical values and then scaled according to the LSB as it is written in the ICD. Values are recorded into a struct which is passed by reference to the main loop which calls the Read function.

- FACE::NOT_AVAILABLE → no data are available on the serial connection, so no update for coordinate is possible and last info is kept constant. If it is not already running, a timer is activated to measure the time without receiving data and if a threshold is overcome, by means of the Device Protocol Mediation (subsection 4.2.3.3) a copy of the missing data in retrieved by FTP: indeed, the EGI establish a new serial connection with the DPM UoC and reads data from another serial connection by means of the same IOSS APIs for the normal reading. Thus, for the EGI, the operation to retrieve data is made through the same mechanism, while only the data source changes.

4.2.3.2.4 EGI$_{m}anager$ :: *Loop*

```
int EGI_manager::Loop(){

    ...

    while(1){
        memset((void*) &msg, 0, sizeof(messaggio_EGI_t));
        memset((void*) &mnt_msg, 0, sizeof(monitor_packet_t));
        myEGI.Read(&msg, status);
        switch(status){
            case FACE::NO_ERROR:
                to_cnt = 0;
                msg_TS.speed = msg.ac_speed;

                msg_TS.speed_NED.X = msg.speed.X;
                msg_TS.speed_NED.Y = msg.speed.Y;
                msg_TS.speed_NED.Z = msg.speed.Z;

                msg_TS.pos.lon = msg.pos.lon;
                msg_TS.pos.lat = msg.pos.lat;
                msg_TS.pos.alt = msg.pos.alt;

                msg_TS.att.roll = msg.att.roll;
                msg_TS.att.pitch = msg.att.pitch;
                msg_TS.att.yaw = msg.att.yaw;
                msg_TS.att.heading = msg.att.hd;

                msg_TS.acc_NED.X = msg.acc.X;
                msg_TS.acc_NED.Y = msg.acc.Y;
                msg_TS.acc_NED.Z = msg.acc.Z;
                msg_TS.timestamp = msg.tmp;

                FACE::TS::Send_Message(
                        connId, FACE::INF_TIME_VALUE, txn, msg_TS,
                        max_msg_size, status);


                if (status != FACE::NO_ERROR) {
                    perror("EGI_m=>Impossible to send the message!");
                    return static_cast<int>(status);
                }

                msg_old = msg;

    ...
        }
    }

}
```

The EGI_manager Class, which is structured as a Singleton in order to prohibit to have any copy, embodies a EGI Class object. The latter is used to read from the serial and the former is use to communicate via TSS interface with the PCS A/C Position application. The code excerpt here reported, shows the infinite loop which is run in order to continuously read from serial connection (as explained in subsubsection 4.2.3.2.3) and send the message to A/C Position application via TSS interface. In case of FACE::NO_ERROR, the message read from serial is copied field by field into a packet with a structure designed according to the respective topic. On the contrary, in case of FACE::TIMED_OUT, a counter is used to keep track of how many packets have been lost (not reported here in the code, see Appendix A A). In a first moment, since an accidental lost can happen, the last valid packet is sent without signalling the failure, but when the number of missing packets keep

increasing beyond a certain threshold, the failure is signalled by means of a zero timestamp. In the last part of the loop (not reported here in the code, see Appendix A A), every second, a message is sent via IOSS Serial implementation to the PSCS Health Manager in order to update it about the status of the process.

4.2.3.2.5 EGI::main.cpp

```
#include "EGI.h"
#include "EGI_manager.h"


void signal_handler(int signal, siginfo_t *info, void *extra){
    if(signal==SIGKILL){
        printf("SIGKILL received!!!\n");
        ...
        EGI_manager::getMyEGI()->Destroy_TS_Connection();
        EGI_manager::getMyEGI()->Flush();
        EGI_manager::getMyEGI()->Close();
    }
    else{
        perror("Another unknown signal detected!");
    }
}

int main() {

    struct sigaction sa;
    sa.sa_sigaction = signal_handler;
    sa.sa_flags = SA_SIGINFO;
    sigemptyset(&sa.sa_mask);
    sigaction(SIGKILL, &sa, NULL);

    EGI_manager::getMyEGI()->Initialize();
    EGI_manager::getMyEGI()->Loop();
    EGI_manager::getMyEGI()->Destroy_TS_Connection();
    EGI_manager::getMyEGI()->Flush();
    EGI_manager::getMyEGI()->Close();

    return EXIT_SUCCESS;
}
```

At the very beginning of the function, the signal handler is instantiated in order to manage, upon the `SIGKILL` signal reception, the closing of all connections and the deallocation of memory structures. In the main function, the signal handler struct is created and assigned with proper values; then, the unique EGI_manager Singleton object is allocated, initialized and its method *Loop()* is called to start reading from serial and publishing messages via TSS APIs.

4.2.3.3 DPM

The Device Protocol Mediation (DPM) is a part of the PSSS which allows for protocol conversion between different UoCs. In the current implementation under analysis, DPM is used to recover by FTP (File Transfer Protocol) missing data in the case the serial connection between the sensors and the managing PSSS apps fails. Indeed, if either the EGI or the GPS detects a failure in the serial connection, it can retrieve a backup copy of the sensor data from a server by reading over the serial. The DPM is structured as a process in which two threads are both waiting for establishing a

connection with the respective failing module of the PSSS (i.e. the EGI or GPS). Indeed every thread is stuck on a blocking read waiting for a signal by the other end: when the desired packet from the PSSS module arrives, it release the DPM thread which starts recovering, by means of proper function wrapping POSIX sockets, a database file in which last second positions of the simulator are serialized. Then, each record is read and transmitted over the serial connection to the PSSS failing UoC. In the following, the read loop, in which the backup structure is first retrieved from file and then sent record by record to the other PSSS UoC, is reported.

```
...
/*
 * Serial configuration and inizialization
 */
while(1){

        serial.Read(handle, FACE::IOSS::Serial::WAIT_FOREVER, &rd, 0,
    return_code);  //blocking

        retrieve_file_from_server();
        int fd = open("backup.txt", O_RDONLY);

        backup_t bak;
        read(fd, (void*) &bak, sizeof(bak));

        uint16_t i=0;
        while(i<bak.idx){

            std::string c1_bin = numericToBinary<double>(bak.pos[i].lat/
    MAX_LAT_LON, 32);
            ...

            serial.Write(handle, FACE::IOSS::Serial::WAIT_FOREVER, &wr,
    PACKET_SIZE, return_code);
            i++;
        }

        close(fd);
    }
```

Each record of the *bak* structure contains a struct instance in which all the aircraft data are stored: each record is read in a loop and sent via serial to the PSSS module in charge of the failing sensor.

4.2.3.4 Health Manager

In order to have a clear surveillance of the behaviour of the ongoing applications, another software component is needed: the System Level Health Monitoring inside the PSCS, here called Health Manager. This is an application assumed to be fail-safe whose task is to check whether the other applications are correctly working or are encountering a problem or are stuck at some point. This HM collects every second a status message from the EGI and GPS by serial communication and from the A/C Position application by Transport Service capability. Upon starting, the HM app launches all the other software components and records the process identifiers, the *pids*. Then, in a infinite loop, every each second, waits for the messages from EGI, GPS and A/C Position UoP to be delivered setting a timeout of one second: if one of the messages fails to arrive, the HM increases a counter to keep trace of how many

messages from each app have been lost. If more than ten messages in a row from one app are lost, that app is considered to be stuck and this is signalled to the cockpit window by means of sockets. Then, inside the window, some buttons allow to react to the HM and ask for the app to be killed and launched again by means of the sequence of *fork* and *execle*. In the following two excerpts of the code are reported

```
    if((processes[1]=fork())==0){
        printf("Launching DF...\n");
        fflush(stdout);
        execle("path/to/DF",
               nullptr);
    }
```

In the previous code the HM process splits itself by fork and the child process is replaced with the A/C Position application process image by means of *execle* system call. The new process *pid* is collected in an array in order to be used in case the process gets stuck and needs to be killed.

The following code has the goal to check the behaviour of the other PSSS and PCS apps by exchanging messages with them. The algorithm can be divided into three parts. The first is a message exchange by means of IOSS serial connection or by TS publisher-subscriber mechanism. In the second part, the program verifies that all the messages are received, and, if not, which one is missing and for how long. If an app is not sending message for more than ten seconds, the app is assumed to have crashed and gets killed. In the last part, a file, in which the *pids* are recorded, is updated with the new process id.

```
while(1){

        isSomeKilled = false;
        cnt = 0;

        FACE::TS::Receive_Message(
                connId_mnt, 1e9, txn_mnt, msg_TS_monitor,
                max_msg_size_mnt, status_monitor);
        if (status != FACE::NO_ERROR) {
            lost[0]++;
        }
        else{
            cnt++;
            lost[0] = 0;
        }

        ser_egi.Read(handle_egi, 1e9, &rd, MONITOR_PACKET_SIZE, status);
        if (status != FACE::NO_ERROR) {
            lost[1]++;
        }
        else{
            cnt++;
            lost[1] = 0;
        }

        ser_gps.Read(handle_gps, 1e9, &rd, MONITOR_PACKET_SIZE, status);
        if (status != FACE::NO_ERROR) {
            lost[2]++;
        }
        else{
            cnt++;
            lost[2] = 0;
        }

        // HM CHECKING
        if(cnt==3){
```

```
37              printf("HM: all three packets received!!\n");
38          }
39          else{
40              perror("Message lost...");
41              if(lost[0]>10){ //DF
42                  perror("DF stopped working...");
43                  // kill & restart the process
44                  kill(processes[1], SIGKILL);
45
46                  pid_t pid;
47                  if((pid=fork())==0){//child
48                      execle("path/to/DF", NULL);
49                      perror("");
50                  }
51                  else{ //father continues
52                      processes[1] = pid;
53                      isSomeKilled = true;
54                  }
55
56
57              }
58              else if(lost[1]>10){  //EGI
59                  // launch EGI again by means of fork+execle
60              }
61              else if(lost[2]>10){  //GPS
62                  // launch GPS again by means of fork+execle
63              }
64          }
65
66          if(isSomeKilled){
67              // update pids file
68          }
69
70      }
```

An important observation is needed at this point: such processes can be killed in this straightforward manner since they don't possess any internal state which need to be saved before being flushed from the memory. These processes simply receive and send packets back. If they are launched mid-operation, only a few packets are lost in the few milliseconds it takes to start them, so there's no significant impact on performance.

4.2.3.5 PSGS Component

This graphical part, residing inside the Platform-Specific Graphic Segment, has been realized with Qt, a framework used to develop multi-platform GUI. On this main display, all the previously discussed pieces of info, such as A/C speed, heading, attitude, velocities and accelerations, are showed in real time, upon receiving via TSS from the PCS A/C Position UoP. The main window is divided vertically into two panels: the one on the left, the `AIRCRAFT DATA & A/C 2D MAP` panel, represents the aircraft on a 2D map providing also the info about latitude, longitude, altitude, aircraft speed and heading while the panel on the right, the `ATTITUDE DATA` panel, provides the info about pitch, roll and yaw by means of an artificial horizon (pitch and roll) or a text box (yaw). The following image (Figure 4.5) shows the aspect of the window and all the discussed widgets. From the communication point of view, the application is equipped with a special class *MessageReceiver* which is in charge of polling for messages on the topic onto which the A/C Position Application publishes

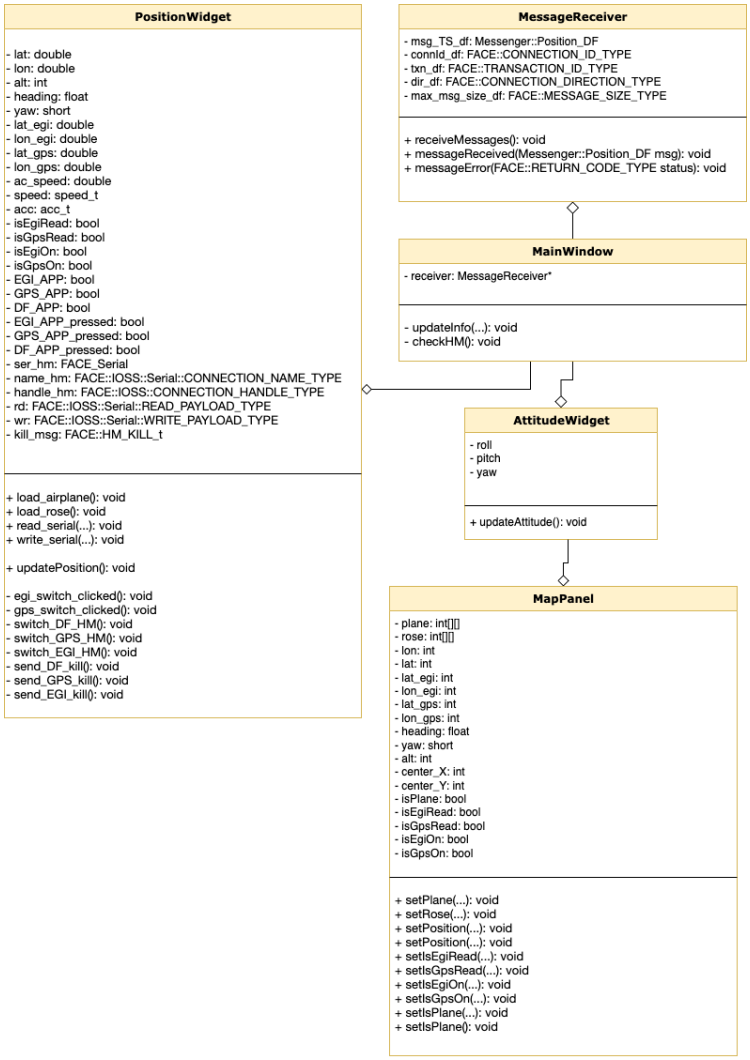

Figure 4.4: PSGS application UML

the fused data. In Figure 4.4 is reported the application UML to help visualize the software component structure.

4.2.3.5.1 messagereceiver.cpp

```
#include "messagereceiver.h"

MessageReceiver::MessageReceiver(QObject *parent)
    : QObject(parent)
{

    FACE::TS::Initialize("FACE_LAYERS/TSS/TS/config_static_PSGS.ini", status);

    FACE::TS::Create_Connection(
        "sub_DF", FACE::PUB_SUB, connId_df, dir_df,
        max_msg_size_df, FACE::INF_TIME_VALUE, status);

    qRegisterMetaType<Messenger::Position_DF>("Messenger::Position_DF");

    ...
}


void MessageReceiver::receiveMessages() {
```

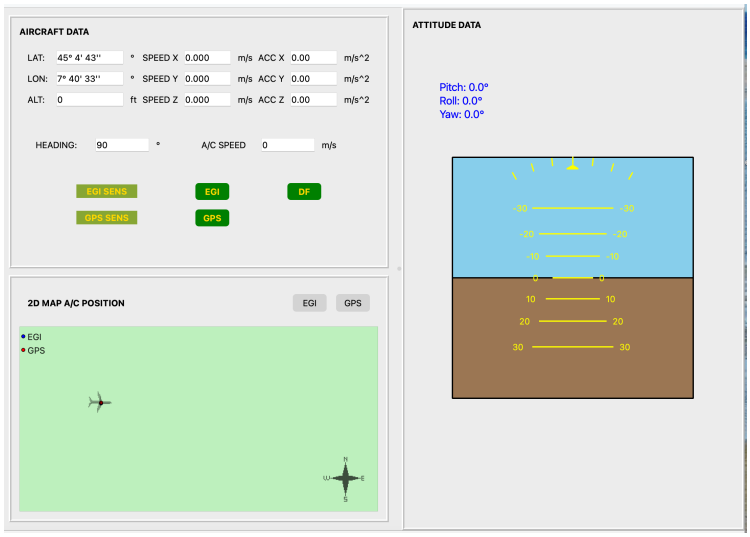

Figure 4.5: PSGS UoC, the graphical component to show the data about the aircraft

```
20      while (true) {
21          FACE::TS::Receive_Message(connId_df, 1.5e8, txn_df, msg_TS_df,
        max_msg_size_df, status);
22          printf("Msg received\n");
23          if (status == FACE::NO_ERROR) {
24              emit messageReceived(msg_TS_df);
25          }
26          else if(status==FACE::INVALID_CONFIG){
27
28              FACE::TS::Initialize("FACE_LAYERS/TSS/TS/config_static_PSGS.ini",
        status);
29              FACE::TS::Create_Connection(
30                  "sub_DF", FACE::PUB_SUB, connId_df, dir_df,
31                  max_msg_size_df, FACE::INF_TIME_VALUE, status);
32
33          }else {
34              emit messageError(status);
35          }
36          usleep(1e4);
37      }
38  }
39
40  MessageReceiver::~MessageReceiver(){
41      FACE::TS::Destroy_Connection(connId_df, status);
42  }
```

As mentioned before, the application exploits the TSS to receive information from the PCS A/C Position application. In other words, this application uses the APIs provided by the FACE TSS implementation of OpenDDS to retrieve messages. Thus, in a continuous loop, paused regularly of 10ms, this class calls the *Receive_Message()* function of the TSS interface: in a positive case, a packet is retrieved, while in the opposite case an error is returned and, in case on *INVALID_CONFIG*, initialization and subscriber instantiation is repeated. An aspect worth noticing is that the integration of Qt with the OpenDDS FACE TSS implementation requires that the message type, associated to a topic and defined in an IDL file, must be registered to the Qt application (line 13). This allows the application to deal with the new type as in the case of the *emit* signal function: this method, whose technical definition is

signal, allows to send data structure from a component to another inside the graphical application.

4.2.3.5.2 mainwindow.cpp

```
1
2  MainWindow::MainWindow(QWidget *parent)
3      : QMainWindow(parent)
4      , ui(new Ui::MainWindow)
5  {
6      ...
7
8      connect(thread, &QThread::started, receiver, &MessageReceiver::
       receiveMessages);
9      connect(receiver, &MessageReceiver::messageReceived, this, &MainWindow::
       updateInfo);
10
11     ...
12 }
13
14
15 void MainWindow::updateInfo(Messenger::Position_DF msg_TS_df){
16     switch(msg_TS_df.fuse_src){
17         case FACE::FUSION_TYPE::BOTH:{
18             //printf("BOTH\n");
19             fflush(stdout);
20             double lat = msg_TS_df.pos.lat;
21             double lon = msg_TS_df.pos.lon;
22             int alt = msg_TS_df.pos.alt;
23
24             float heading = msg_TS_df.att.heading;
25             short pitch = msg_TS_df.att.pitch;
26             short roll = msg_TS_df.att.roll;
27             short yaw = msg_TS_df.att.yaw;
28
29             double ac_speed = msg_TS_df.speed;
30
31             speed_t speed;
32             acc_t acc;
33
34             speed.X = msg_TS_df.speed_NED.X;
35             speed.Y = msg_TS_df.speed_NED.Y;
36             speed.Z = msg_TS_df.speed_NED.Z;
37
38             acc.X = msg_TS_df.acc_NED.X;
39             acc.Y = msg_TS_df.acc_NED.Y;
40             acc.Z = msg_TS_df.acc_NED.Z;
41
42             ...
43 }
```

In the previous code snippet, it is possible to see how data is retrieved by the PSGS display application: once the message is emitted by *MessageReceiver::receiveMessages()* function, as described before, it is intercepted by *MainWindow::updateInfo(...)* method. Indeed, the *Messenger::Position_DF msg_TS_df* unique argument, inside the function signature, is the very same packet sent by the *MessageReceiver* object, which, in turn, was retrieved via TSS. As shown in the class constructor at the very beginning of the code, this is made possible by creating a thread on which *MessageReceiver::receiveMessages()* will run (line 8) probing for messages and, once a message is received, the *messageReceived* signal is emitted to deliver the packet to *updateInfo* function since the signal and the function have been connected in line 9.

Moving forward in the code, it is possible to understand how values are passed from TSS packet to the graphical component in order to be shown on the screen.

4.2.3.5.3 Position Panel

The position panel occupies the left half of the window, with an upper section displaying grids of labels and text boxes for latitude, longitude, altitude, plane speed (in NED frame), and plane acceleration. Below this, the aircraft speed and heading are shown on a single grid layout line. At the bottom, there's a set of LEDs and buttons organized in a grid, including two labels indicating the EGI and GPS sensor status and three buttons to manage the sensor and A/C position applications. These buttons display the app status and allow the user to send a kill signal to the Health Manager app, which will restart any failing application if necessary. In the following an extract of the code is reported in order to show the functioning of the app management buttons on the pure graphical side; only the code for A/C Position button is showed, since the EGI and GPS functioning is the very same.

```
1
2       //DF
3       if(!DF_APP_pressed && DF_APP){ //not pressed & app ok
4           btn_DF->setStyleSheet(
5               "background-color: green; "
6               "color: gold; "
7               "font-weight: bold; "
8               "border-radius: 5px; "
9               "padding: 7px;"
10              );
11      }
12      else if(DF_APP_pressed && DF_APP){ //pressed & app ok
13          btn_DF->setStyleSheet(
14              "background-color: darkgreen; "
15              "color: gold; "
16              "font-weight: bold; "
17              "border-radius: 5px; "
18              "padding: 7px;"
19              );
20      }
21      else if(!DF_APP_pressed && !DF_APP){ //not pressed & app not ok
22          btn_DF->setStyleSheet(
23              "background-color: red; "
24              "color: gold; "
25              "font-weight: bold; "
26              "border-radius: 5px; "
27              "padding: 7px;"
28              );
29      }
30      else{ // DF_APP_pressed && !DF_APP //pressed & app not ok
31          btn_DF->setStyleSheet(
32              "background-color: darkred; "
33              "color: gold; "
34              "font-weight: bold; "
35              "border-radius: 5px; "
36              "padding: 7px;"
37              );
38      }
```

4.2.3.5.4 Map Panel

This panel is nothing but a big light green rectangle in which a silhouette of a plane is represented. Near the panel title, two buttons are inserted in order to enable or disable the depiction of the position given by the EGI or GPS sensor: this

allows to show only the datum given by one sensor hiding the other, demonstrating that each sensor provides different info because of the noise and then these info are merged together to obtain the final position of the plane. The drawing of the plane is obtained starting from a square binary matrix in which every cell represents a point of the image: 0 if no point of the plane is to be represented, 1 otherwise. This matrix is scanned each time in order to draw the plane and for each of these points a translation and a rotation is applied: since the $(0;0)$ coordinate of the matrix is on the top left but the rotation is about z axis which is in the center of mass of the plane (the center of the matrix), all the $(i;j)$ points must be translated back by a quantity equal to the position of the center of the matrix represented by the point $p_0(C_x;C_y)$; not doing so the following rotation would be about an axis passing in $(0;0)$ and not in the center. The rotation is represented by a 2x2 rotation matrix whose rotation angle is the A/C heading. After the rotation is completed, the center position is adjusted by adding an offset to reposition the translated airplane drawing. Additionally, the coordinates (expressed in pixels) are also included. The plane is depicted in black with a bit of transparency in order to show the underlying two dots representing the sheer info from both sensors without data fusion while the plane itself is the merged position. In the following the mathematical definition of the rotation matrix is reported along with the two equations used to compute the position of the aircraft.

$$M = \begin{bmatrix} cos(\psi) & -sin(\psi) \\ sin(\psi) & cos(\psi) \end{bmatrix}$$

$$\hat{p} = M(p - p_0) \tag{4.1}$$

$$p_f = \hat{p} + p_0 + airplane\_coordinates \tag{4.2}$$

The *airplane_coordinates* used in 4.2 are the rescaled latitude and longitude. Inside the map area, on the screen only a certain portion of latitudinal and longitudinal space can be represented; once the coordinates are received, in order to paint the plane, they must be rescaled by means of the following equations.

$$\begin{aligned} airplane\_coordinates_x &= (lon - lon_0) * size_x/(2 * \Delta_x) \\ airplane\_coordinates_y &= (lat - lat_0) * size_y/(2 * \Delta_y) \end{aligned} \tag{4.3}$$

where $size_x$ and $size_y$ are the specific dimensions of the map area in pixel and here's why this application is located in the Platform-Specific segment since this values are specific to the hardware used. Δ_x and Δ_y are the amount of vertical and horizontal space representable on the map from a central point. lon_0 and lat_0 are respectively the minimum longitude and latitude representable corresponding to the top left corner of the map, while *lon* and *lat* are the data retrieved by the A/C Position application via TSS.

Beside the plane drawing, coordinates retrieved in the EGI and GPS packets without operating data fusion are displayed as coloured dots in order to have the user appreciate the difference between the original noisy data and the merged data

represented by the plane.

```
    painter.setBrush(Qt::green);
    painter.setOpacity(0.2);
    painter.drawRect(0, 0, this->width(), this->height());


    if(isEgiRead && isEgiOn){
        painter.setBrush(Qt::blue);
        painter.drawEllipse(Cx+lon_egi-2, Cy+lat_egi-2, 5, 5);
    }

    if(isGpsRead && isGpsOn){
        painter.setBrush(Qt::red);
        painter.drawEllipse(Cx+lon_gps-2, Cy+lat_gps-2, 5, 5);
    }

    if(isPlane){

        painter.setBrush(Qt::darkGray);
        painter.setOpacity(0.4);
        double M[2][2];
        double head_d = (double)heading;

        M[0][0] = cos(head_d);
        M[1][1] = cos(head_d);
        M[0][1] = -sin(head_d);
        M[1][0] = sin(head_d);

        int i,j;
        for(i=0; i<35; i++){
            for(j=0; j<33; j++){
                if(plane[i][j]==1){

                    //centered coordinates
                    int y = i-Cy;
                    int x = j-Cx;

                    //rotated point
                    double x_n = M[0][0]*(double)x + M[0][1]*(double)y;
                    double y_n = M[1][0]*(double)x + M[1][1]*(double)y;

                    // back to orginal coordinates
                    x = (int)x_n+Cx+this->lon;
                    y = (int)y_n+Cy+this->lat;

                    painter.drawPoint(x,y);
                }
            }
        }
    }
```

In the bottom right corner a wind rose is depicted in order to show the direction to which the plane is moving. This rose is displayed through the same technique used for the plane, with the difference that in this case no rotation is needed so the binary matrix gets read and converted into pixels drawn on the map.

4.2.3.5.5 Attitude Panel

The attitude panel shows the attitude angles of the aircraft: the angles are both written as text in the top left corner of the panel and shown by the artificial horizon. This widget is a square in which the line of the horizon is represented dividing ground and sky: while pitching the line moves vertically, while rolling the line rotates around an axis passing through the vertical centre of the square and both movements can be done at the same time. On the foreground of the widget, as a superimposed

layer, there is the set of horizontal lines to measure the pitch angle and little dashes around an imaginary circumference to represent the roll angle. In addition, while rolling, a small yellow triangle works as an indicator of roll angle and points to the corresponding oblique dash. In the following a simplified version of the code, which doesn't keep into account some graphical refinements, is reported. In the first part the reader could understand how ground and sky are drawn while in the second how the two systems of indicators are depicted.

```
    QSize s = this->size();
    int radius = s.height() / 4;
    int center_x = s.width() / 2;
    int center_y = s.height() / 2;

    double roll_d = roll * M_PI / 180.0;
    double pitch_d = pitch * M_PI / 180.0;
    float slope = tan(roll_d);
    int yOffset = static_cast<int>(center_y + tan(pitch_d) * radius);

    painter.setPen(QPen(Qt::black, 2));
    painter.drawEllipse(QPoint(center_x, center_y), radius, radius);

    QPainterPath skyPath, groundPath;

    int y1 = yOffset + slope * (-radius);
    int y2 = yOffset + slope * radius;

    // Draw sky and ground
    skyPath.moveTo(center_x - radius, center_y - radius);
    groundPath.moveTo(center_x - radius, center_y + radius);

    painter.setBrush(QColor(135, 206, 235));
    painter.drawPath(skyPath);
    painter.setBrush(QColor(155, 118, 83));
    painter.drawPath(groundPath);

    painter.setPen(QPen(Qt::black, 2));
    painter.drawLine(QPoint(center_x - radius, y1), QPoint(center_x + radius,
    y2));

    // Draw pitch angle indicator
    painter.setPen(QPen(Qt::yellow, 2));
    for (int i = -30; i <= 30; i += 10) {
        int yLineOffset = static_cast<int>(center_y + tan(i * M_PI / 180.0) *
    radius);
        if (yLineOffset >= center_y - radius && yLineOffset <= center_y +
    radius) {
            painter.drawLine(QPoint(center_x - radius / 2 + (60 - abs(i)),
    yLineOffset),
                             QPoint(center_x + radius / 2 - (60 - abs(i)),
    yLineOffset));
            painter.drawText(QPoint(center_x - radius / 2 + (30 - abs(i)),
    yLineOffset + 5), QString::number(i));
            painter.drawText(QPoint(center_x + radius / 2 - (50 - abs(i)),
    yLineOffset + 5), QString::number(i));
        }
    }

    // Draw roll angle indicator
    for (int i = -30; i <= 30; i += 10) {
        double angle_rad = (i) * M_PI / 180.0 - M_PI / 2;
        painter.drawLine(static_cast<int>(center_x - radius * cos(angle_rad)),
                         static_cast<int>(center_y + radius * sin(angle_rad)),
                         static_cast<int>(center_x - (radius - 10) * cos(
    angle_rad)),
                         static_cast<int>(center_y + (radius - 10) * sin(
    angle_rad));
        }

    double angle_rad = (roll - 3) * M_PI / 180.0 - M_PI / 2;
    QPoint point1(center_x + (radius - 17) * cos(angle_rad), center_y + (
    radius - 17) * sin(angle_rad));
```

```
54      // Same for point2 and point3
55
56      QPoint points[3] = { point1, point3, point2 };
57      painter.setBrush(Qt::yellow);
58      painter.drawConvexPolygon(points, 3);
```

4.2.4 IOSS Implementation

This subsection provides a detailed explanation of the use of IO in this case study. Strength points for the I/O Service are clarity, consistency and extensibility. Clarity is provided by defining a separate I/O Service for each supported I/O bus architecture. Consistency is provided by supporting multiple I/O bus architectures that follow a consistent declaration pattern for an I/O Service. Extensibility is addressed by means of the capability of extending the configuration and controlling declarations for an I/O Service [1]. Each interface follows a consistent pattern to be used with the same syntax for common functions. An instance of an I/O Service represents a particular device bus in the system and the type of this instance is the device bus architecture under consideration.

The IOSS has been realized as a set of header and corresponding implementations, but for commodity, all the code of this segment has been converted into a dynamic library to be linked to executables from other segments without including dozens of header files. The following code snippet, extracted from the PSSS EGI application CMakeLists.txt file, shows how the inclusion of the dynamic library is made.

```
1       add_executable(EGI main.cpp EGI.cpp EGI_manager.cpp ${FACE_TS_SRC})
2       target_link_libraries(EGI ${opendds_libs} path/to/libFACE_Serial.dylib)
```

Remind that a dynamic library reduces executable size and memory usage by allowing multiple programs to share common code at runtime; in other words, instead of including the header and code inside each executable with the effect of having very big executables, the shared library is brought into memory only once and linked at run time if needed, saving time and space. The library has been compiled for both architectures in order to be used both on ARM and on x86_64 devices.

```
1       set(CMAKE_OSX_ARCHITECTURES "arm64;x86_64")
2       ...
3       add_library(FACE_Serial SHARED ${FACE_SERIAL_SOURCES})
```

In the current software implementation, several serial connections are established: between the sensors simulators and PSSS managing apps or between these PSSS managing apps and the PSCS Health Manager app.

4.2.4.1 Serial

The FACE Technical Standard Edition 3.1 [1] provides a pretty detailed example for the development of the code to be used in this segment and the correlated functions

and behaviours. Unfortunately lots of the code is missing with the purpose of giving the software integrator the freedom of writing the necessary details for the specific application. For what regards the *Read* and *Write* functions, no signature or example is provided: instead, only guidelines are given with the goal of describing the overall behaviour and the returned values expected by the rest of the environment. One of the first guideline is about the timing of these functions and about the aspect of whether being blocking or not since three different possibilities are contemplated: blocking, not blocking and with timeout. In the first case, the function waits for data to be available in the case of *Read*, or for a buffer to be empty in the case of *Write*. In the case of not blocking at all, both functions attempt to either read or write and then return with a different status code to signal whether they have been able to accomplice their task or not. The last case, the one with a timeout, both functions try to perform their task within the requested time and, after that, they return a code indicating whether the action succeeded or not. The different behaviour is possible by acting on parameters of the serial port represented by some numeric options of the file descriptor representing the serial port. As explained in the previous chapter, in the code development for the thesis, serial hardware has been simulated inside the terminal of the personal computer, and in order to open a serial connection, the following command must be typed:

```
socat -d -d pty,raw,echo=0 pty,raw,echo=0 2
```

The visualized output on the console is something like:

```
2024/10/27 19:23:58 socat[8947] N PTY is /dev/ttys002
2024/10/27 19:23:58 socat[8947] N PTY is /dev/ttys003
```

In the current example, the strings `/dev/ttys002` and `/dev/ttys003` are two file paths for the two serial endpoints to pass to *fopen()* or *open()* system call in order to retrieve the file descriptors to be used in the serial operations. For every serial communication involved in the software implementation, a *socat* command is needed and the related file paths are written into a `cfg` which contains all the parameters to set at the opening of the serial connections, such as baud rate or parity bits.

The Serial is implemented as a class which uses other helping functions contained in other headers. In the following part, the header is first presented and commented. After that, a detailed explanation of each major function is provided.

4.2.4.1.1 FACE_Serial.h

```

#include ...

class FACE_Serial{
public:
    FACE_Serial(string devName);
```

```
7       FACE_Serial();
8       virtual ~FACE_Serial();
9
10      /*
11      * public methods and object attributes
12      * ...
13      */
14
15
16  private:
17      long baudRate;
18      short dataBits;
19      short stopBits;
20      short parity;
21      short flowControl;
22      bool echo;
23      bool translateLF;
24      bool translateEchoCR;
25      char deviceName[MAX_LINE_LENGTH];
26      char *name;
27      int fd_m;
28
29      ConfigFile *configFile = NULL;
30      MyCallback *my_cb = NULL;
31  };
```

In the first part of the header, constructor, destructor and the essential function for the serial class are defined as public. This is followed by a block of getters for the private fields of the class which are listed in the end of the header. In the very end `ConfigFile` and `MyCallback` objects are declared in order to contain, respectively, the configuration file with all the parameters for the serial connection and the callback pointer.

4.2.4.1.2 mySerial::Initialize

```
1   void FACE_Serial::Initialize(
2       const FACE::CONFIGURATION_RESOURCE& config_resource,
3       FACE::RETURN_CODE_TYPE& return_code) {
4
5       baudRate=115200; dataBits=8; stopBits=1; parity=0; flowControl=0;
6       echo=translateLF=translateEchoCR=false;
7       fd_m = -1;
8       for(short i=0; i<MAX_CONNECTIONS; i++){
9           callback_vector[i] = nullptr;
10      }
11
12      if(configFile != NULL){
13          ...
14          configFile->Open(fullName, mySection, return_code);
15          if(return_code == FACE::NO_ERROR){
16              ...
17              configFile->Read(mySection,"deviceName",static_cast<void *>(buffer
        ),
18                               sizeof(buffer), sz, return_code);
19              ...
20              configFile->Read(mySection,"baudRate",static_cast<void *>(buffer),
21                               sizeof(buffer),sz,return_code);
22              int brate=atoi(buffer);
23              if(brate >=50 && brate <=230400) baudRate=brate;
24              ...
25              configFile->Close(mySection, return_code);
26              return_code=FACE::RETURN_CODE_TYPE::NO_ERROR;
27          } ...
28  }
```

This function has the goal to first set default values to bus parameters and then read them from a configuration file and check whether the values are acceptable or not: if yes, the parameters are overwritten with the new values, otherwise default values are kept. If the configuration file is not reachable in any way, all the initialization process is discarded, an error is reported and no serial connection can be established or used.

4.2.4.1.3 mySerial::Open_Connection

```
void FACE_Serial::Open_Connection(
        const FACE::IOSS::Serial::CONNECTION_NAME_TYPE& name,
        FACE::TIMEOUT_TYPE timeout,
        FACE::IOSS::CONNECTION_HANDLE_TYPE& handle,
        FACE::RETURN_CODE_TYPE& return_code) {
    ...
    fd_m = open(deviceName, O_RDWR | O_NOCTTY | O_SYNC);
    if (fd_m < 0) {
        fprintf(stderr,"FACE_Serial failed to open serial device: %s. errno=%d
    \n",
                deviceName,errno);
        return_code=FACE::RETURN_CODE_TYPE::INVALID_CONFIG;
    } else {
        fprintf(stderr,"DEBUG: Serial port %s opened at speed %ld\n",
    deviceName, baudRate);
        handle = fd_m;
        valid_handles[conn_idx] = handle;
        conn_idx++;
        conn_idx%=MAX_CONNECTIONS;
    }
    ...
}
```

Once all the parameters have been set, the connection can be opened by means of the *open*() system call provided by the POSIX APIs. The `deviceName` is retrieved from the configuration file and it's used as pathname for the connection; the file descriptor returned while opening the connection is used as handle by the caller for future serial use. Furthermore, the handle is also saved in a special vector (`valid_handles[...]`) in order to be available for future check on its validity.

4.2.4.1.4 mySerial::Read

```
void FACE_Serial::Read(
        FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
        FACE::TIMEOUT_TYPE timeout,
        FACE::IOSS::Serial::READ_PAYLOAD_TYPE *rd,
        FACE::UnsignedLong effective_payload_size,
        FACE::RETURN_CODE_TYPE& return_code){

    return_code = FACE::NO_ERROR;
    ... check whether in parameters are ok

    struct termios options;
    memset(&options, 0, sizeof(options));

    // Retrieve current parameters for the serial port
    if (tcgetattr(handle, &options) != 0) {
        perror("Error in reading!!");
        close(handle);
        return_code = FACE::INVALID_CONFIG;
        return;
    }
```

```
21
22      options.c_cc[VMIN] = 0;
23      if(timeout==FACE::IOSS::Serial::WAIT_FOREVER or timeout==FACE::
        INF_TIME_VALUE){ //blocking
24          options.c_cc[VTIME] = 0;
25          options.c_cc[VMIN] = 1;
26      }
27      else if(timeout==FACE::IOSS::Serial::NO_WAIT){
28          options.c_cc[VTIME] = 0;
29      }
30      else{
31          options.c_cc[VTIME] = timeout/1e8; // from nsec to dsec
32      }
33
34      // set params to serial port
35      if (tcsetattr(handle, TCSANOW, &options) != 0) {
36          perror("Error in configuration!!");
37          close(handle);
38          return_code = FACE::INVALID_CONFIG;
39          return;
40      }
41
42      ssize_t total_bytes_read = 0;
43      ssize_t bytes_read;
44
45      if(effective_payload_size>0){
46          bytes_read = read(handle, rd->payload, effective_payload_size);
47
48          if(bytes_read>0 && bytes_read<effective_payload_size){
49              return_code = FACE::INVALID_CONFIG;
50              return;
51          }
52
53          if(bytes_read==0){
54              return_code = FACE::TIMED_OUT;
55              return;
56          }
57
58          if(bytes_read==effective_payload_size){
59              struct timespec ts;
60              clock_gettime(CLOCK_MONOTONIC, &ts);
61              rd->received_time = (long long) (ts.tv_nsec)+(ts.tv_sec*1000000000
        LL);
62              return_code = FACE::NO_ERROR;
63              return;
64          }
65      }
66      else{
67          char buffer[DATA_CHUNK_SIZE];
68
69          while(1){ // multiple readings are necessary
70              bytes_read = read(handle, buffer, DATA_CHUNK_SIZE);
71
72              if(bytes_read==0 and total_bytes_read==0){
73                  return_code = FACE::TIMED_OUT;
74                  return;
75              }
76
77              if(bytes_read<DATA_CHUNK_SIZE){
78                  memcpy(rd->payload + total_bytes_read, buffer, bytes_read);
79                  total_bytes_read+=bytes_read;
80                  return_code = FACE::NO_ERROR;
81                  struct timespec ts;
82                  clock_gettime(CLOCK_MONOTONIC, &ts);
83                  rd->received_time = (long long) (ts.tv_nsec)+(ts.tv_sec
        *1000000000LL);
84                  return;
85              }
86
87              if(bytes_read==DATA_CHUNK_SIZE){
88                  memcpy(rd->payload + total_bytes_read, buffer, bytes_read);
89                  total_bytes_read+=bytes_read;
90              }
91          }
92
```

```
93      }
94
95  }
```

In the very beginning of the *Read*, the validity of the timeout and the handle is checked. According to the timeout parameter, the read can be performed in three different ways: FACE::WAIT_FOREVER, FACE::NO_WAIT and with a precise timeout. Thus, the read behaviour can be completely blocking, blocking for a certain amount of time or not blocking at all. The different behaviour can be set by acting on the serial port descriptor with the APIs *tcgetattr* and *tcsetattr* which allow to get and set the attributes describing the parameters of the serial port, thus acting on the options regarding the timer. After this, according to the value of *effective_payload_size*, a single read or multiple reads can occur. Indeed, only one read is performed if the packet can be read all at once since the size of payload is known, or alternatively, multiple reads are necessary in order to probe the actual dimension of the payload since unknown.

4.2.4.1.5 mySerial::Write

```
1  void FACE_Serial::Write(FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
2                          FACE::TIMEOUT_TYPE timeout,
3                          FACE::IOSS::Serial::WRITE_PAYLOAD_TYPE *wr,
4                          FACE::UnsignedLong effective_payload_size,
5                          FACE::RETURN_CODE_TYPE& return_code){
6
7      return_code = FACE::NO_ERROR;
8      ... check whether in parameters are ok
9
10     struct termios options;
11     memset(&options, 0, sizeof(options));
12
13     // Retrieve current parameters for the serial port
14     if (tcgetattr(handle, &options) != 0) {
15         perror("Error in reading!!");
16         close(handle);
17         return_code = FACE::INVALID_CONFIG;
18         return;
19     }
20
21     options.c_cc[VMIN] = 0;
22     if(timeout==FACE::IOSS::Serial::WAIT_FOREVER){ // blocking
23         options.c_cc[VTIME] = 0;
24         options.c_cc[VMIN] = 1;
25     }
26     else if(timeout==FACE::IOSS::Serial::NO_WAIT){ // not blocking
27         options.c_cc[VTIME] = 0;
28     }
29     else{
30         options.c_cc[VTIME] = timeout/1e8; // from nsec to dsec
31     }
32
33     // set params to serial port
34     if (tcsetattr(handle, TCSANOW, &options) != 0) {
35         perror("Error in configuration!!");
36         close(handle);
37         return_code = FACE::INVALID_CONFIG;
38         return;
39     }
40
41     ssize_t bytes_written = 0;
42
```

```
43      if (effective_payload_size == 0) { // effective_payload_size is unknown
44          bytes_written = write(handle, wr->payload, PAYLOAD_DATA_MSG_SIZE);
45          fsync(handle);
46      } else { // effective_payload_size is known
47          bytes_written = write(handle, wr->payload, (size_t)
        effective_payload_size);
48          fsync(handle);
49      }
50
51      ...
52  }
```

In the very beginning of the *Write* the validity of the timeout and the handle is checked. According to the timeout parameter the write can be performed in three different ways: FACE::WAIT_FOREVER, FACE::NO_WAIT and with a precise timeout, meaning that the write behaviour can be completely blocking, blocking for a certain amount of time or not blocking at all. The different behaviour can be set by acting on the serial port descriptor with the APIs *tcgetattr* and *tcsetattr* which allow to get and set the attributes describing the parameters of the serial port, thus acting on the options regarding the timer. After this, according to the value of *effective_payload_size*, if the values is unknown a default size will be written, otherwise the exact dimension will be used. Differently from the *Read*, no multiple writings are necessary since the size of payload is either known or under a maximum threshold and in either case one writing operation is sufficient. Then, the correctness of the action is checked in order to communicate through the *return_code* the outcome of the write call.

4.2.4.1.6 mySerial::Flush

In the following paragraph a very small but important function is presented: *Flush*. This code snippet flushes and clears the serial connection buffer, discarding any undelivered data. This step is essential when closing a connection in an application using the Serial service, as it prevents the transmission of data sent just before closure as they should be discarded. By emptying the channel, it also minimizes lag during intermittent transmission.

```
1  void FACE_Serial::Flush(FACE::IOSS::CONNECTION_HANDLE_TYPE handle, FACE::
       RETURN_CODE_TYPE& return_code){
2      if (tcflush(handle, TCIOFLUSH) != 0) {
3          perror("tcflush error...\n");
4          return_code = FACE::INVALID_CONFIG;
5      }
6      ...
7  }
```

4.2.5 OSS Implementation

The OSS segment contains the Operating System along with Language Run-Time, Application Framework, Health Monitoring (HMFM) and Device Drivers. Indeed, in a proper and real environment, the OS would be a RTOS which is more responsive and more safety critical with respect to a PC Operating System. In the current

work, a minimal version of OSS is implemented since all the software is run on an operating system belonging to POSIX family - actually it is a Linux distribution - and this allows the use of POSIX APIs in order to exploit the functionalities offered by the OS. In a future version of this work, the use of RTOS must be considered in order to guarantee certain performances in terms of real time response, tasks separation/isolation and priority scheduling. Before being implemented in a real case study, of course, the RTOS must be conformant with the FACE Technical Standard and verified to the guidance of DO-178C/ED-12C Design Assurance Level A (DAL A) for Avionics Applications [14, 19].

4.3 Layers External Component

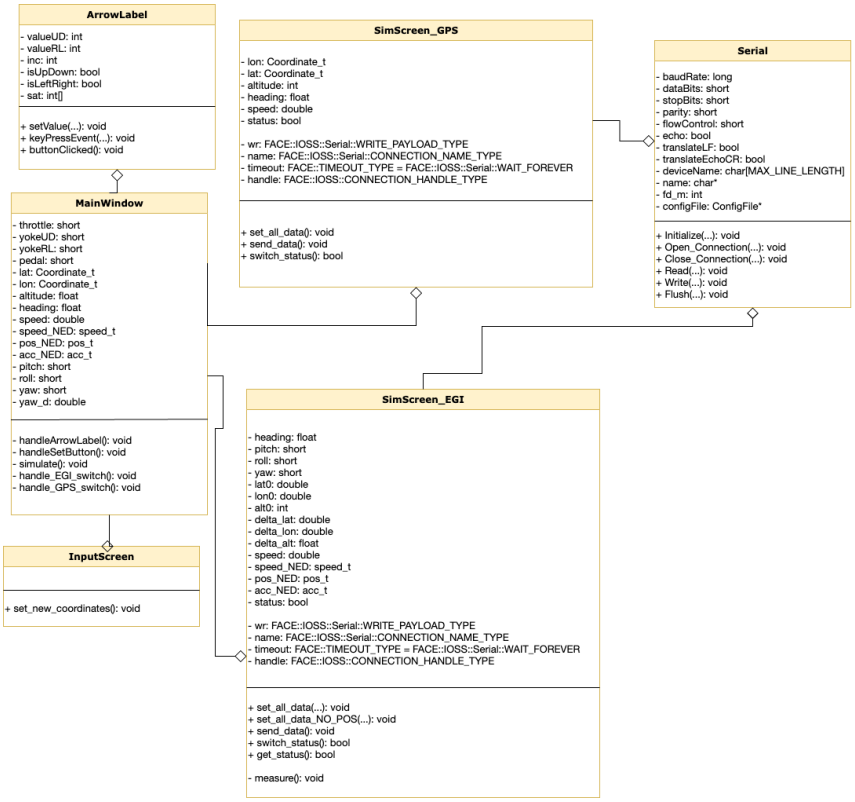

Figure 4.6: Aircraft and sensors simulator UML

4.3.1 Aircraft and Sensors Simulator

This window is designed with the only scope of allowing the user to interact with the FACE software architecture, producing data to pass through the entire stack to check the correct functionalities of every layer from the bottom to the top. The goal of this piece of software is providing data to the rest of the system starting by acting on throttle and yoke values or setting the current position from which to simulate the aircraft movements. Figure 4.6 shows how the classes inside the application are structured and related. The Flight Simulator in embedded in the input window and

writes a series of bytes onto the Serial by means of the I/O Service segment APIs (see section 3.2.4.1).

Figure 4.7: AC_SIM in initial condition before applying any input

The window, as clearly visible in Figure 4.7, is organized in four different panels, two of which are only for simulation output, while the remaining two are for input to the simulator.

4.3.1.1 Aircraft Sensor Panel

The upper part of the window is split into two twin panels which contain similar data: these panels work respectively as simulator for the EGI and GPS hardware sensors mounted on the aircraft and they produce slightly different readings because of an additive noise; in addition the EGI differs from the GPS since it show also inertial data such as speed and acceleration on the three axis. The values, that these sensors are reading, come from a simulation of the dynamical behaviour of the plane responding to throttle and yoke commands. A basic dynamic behavior is simulated to generate the required data, based on the acceleration from the throttle and the pitch/roll angles set by the yoke. This model simplifies many aeronautical and physical details, excluding factors like stall conditions, aircraft stability, moments of inertia, and other complexities. The focus here is only on providing dynamical data to be shown by the main window contained inside the PCS, after crossing the entire data pipeline of FACE segments.

4.3.1.2 Position Input Panel

The second interactive panel allows users to adjust the plane's current position by entering desired latitude, longitude, and altitude values in the corresponding text boxes. Latitudinal and longitudinal data are represented in degrees, minutes and seconds so multiple text boxes are provided to act on each part of the positional information, while altitude is represented in feet as a floating point quantity.

4.3.1.3 Console Commands Panel

The interaction with the throttle and yoke is done by using exclusively the keyboard arrows. The four arrows allow the user to change the aircraft's roll and pitch. The yaw rotation is simulated as a simple rotation around the plane's z-axis acting on pedals with left-right arrows. The heading changes as a consequence of a rolling motion. The up and down arrow keys modify the pitch, consequently adjusting the altitude, which is limited between 0 and 30000 feet. The pitch and roll angles are limited between -30° and +30°. An additional input field, controlled by the up and down arrow keys, manages the throttle, so the acceleration and thus the speed, which is capped between 0 m/s and 1000 m/s (which in kts, knots, a.k.a. nautical miles per hour, is between 0 kts and 1940 kts), without considering stall conditions or other physical constraints.

4.3.1.4 Simulation

Once initial conditions are set and once the simulated aircraft has reached a sufficient speed to move, latitude, longitude and altitude change over time. During simulation, with a sampling time of 15ms these data are passed to EGI and GPS submodules responsible for the data transmission over serial. As said before the display shows the following info:

- latitude → degrees (°) minute (') seconds (")
- longitude → degrees (°) minute (') seconds (")
- altitude → feet

The simulation works in the NED reference system and a conversion to the other representation is needed. After that, noise is added on simulated data of latitude (expressed in semi-circles as double), longitude (expressed in semi-circles as double) and altitude (expressed in feet as floating point) and passed to the EGI and GPS sub-modules. These sub-modules convert the decimal information into binary strings of floating point numbers and write them onto the serial connection. Data, such as latitude, longitude, altitude and so on, are represented on a number of bits describer in the ICD of the specific sensors (3.5.1, 3.5.2): the choice of this amount of bits is a trade of between efficiency in the transmission and accuracy in data representation. Each datum is first divided by the LSB and converted into binary representation, then is sent over serial by means of IOSS APIs.

In the following paragraphs brief extracts from the code are reported.

4.3.1.4.1 mainwindow.cpp

```
1 void MainWindow::simulate(){
2
3     double delta_speed = (MAX_ACCELERATION*(((double)throttle)/100.0)-(
      DRAG_RES*speed/MAX_SPEED))*delta_T_sec;
4     speed += delta_speed;
```

```
5
6       if(speed>MAX_SPEED){
7           speed=MAX_SPEED;
8       }
9       if(speed<0.0){
10          speed=MIN_SPEED;
11      }
12
13      if(speed>0){
14          pitch = yokeUD;
15          roll = yokeRL;
16          yaw_d += pedal/200;
17
18          ...
19
20          // the heading follows an empirical formula which closely approximates
        a real behaviour
21          heading += (float) (deg2rad((roll/20000.0)))+sign(roll)*(speed
       /500000.0)*(delta_T_sec/(1.5e-2));
22          if(heading>(float)2.0*M_PI){
23              heading-=(float)(2.0*M_PI);
24          }
25          if(heading<0.0f){
26              heading+=(float)(2.0*M_PI);
27          }
28
29      }
30
31      // NED COMPONENTS
32      speed_NED.Z = speed*sin(deg2rad(pitch));
33      speed_NED.X = speed*cos(deg2rad(pitch))*cos(heading);
34      speed_NED.Y = speed*cos(deg2rad(pitch))*sin(heading);
35
36      acc_NED.X = (speed_NED.X - old_speed.X)/delta_T_sec;
37      ...
38
39      pos_NED.X += speed_NED.X*delta_T_sec;
40      ...
41
42
43      altitude = (int) (pos_NED.Z*MT2FT);
44
45      if(speed>0){
46
47          // EGI
48          speed_t noisy_speed_NED = speed_NED;
49          noisy_speed_NED.X+= (((double) rand() / RAND_MAX) * 2 * noiseMagnitude
        - noiseMagnitude);
50          ...
51
52          noise = ((double) rand() / RAND_MAX) * 2 * noiseMagnitude -
       noiseMagnitude - 2.0/3600.0;
53          noise_lat = lat+noise;
54          noise_lon = lon+noise;
55          noise = ((double) rand() / RAND_MAX) * 2 * noiseMagnitude_h -
       noiseMagnitude_h + speed/250.0;
56          noise_alt = altitude+(int)noise;
57
58
59          kynematics_t kyn;
60          kyn.speed_NED = noisy_speed_NED;
61          kyn.speed = speed+(((double) rand() / RAND_MAX) * 2 * noiseMagnitude -
        noiseMagnitude);
62          this->s2->set_all_data_NO_POS(kyn, heading, pitch, roll, yaw);
63
64          // GPS
65          /*
66          * Almost same code of EGI without using kinematics_t since no inertial
        behaviour is considered
67          */
68          ...
69      }
70      else{
71          this->s1->set_all_data(lat, lon, altitude, heading, speed);
72
```

```
73          kynematics_t kyn;
74          kyn.speed_NED = speed_NED;
75          kyn.speed = speed;
76          this->s2->set_all_data(kyn, heading, pitch, roll, yaw, c1, c2,
        altitude);
77      }
78
79      ...
80  }
```

This function simulates the behaviour of the plane retrieving the throttle and yoke position: the first is converted into the percentage of the acceleration while the second is summed (rescaled) to the angles of the aircraft in order to project the speed onto the three components of a NED reference system. Then, by means of integrating the speed over time, it is possible to calculate the displacement from the previous NED position and convert it into lat-lon coordinates while altitude is z component of the NED position. In order to perform the sum between the displacement and the coordinates, the latter must be first converted into decimal representation and then the computation can occur. In the case of GPS, some noise is added to the coordinates expressed in radians and then they are sent to both GPS and EGI simulators to be written on the serial connection, since the GPS module work with positional data. In the case of EGI, the noise is added on the speed and no position is computed: these infos are sent to the EGI simulator module, which computes position by integrating speed over time using a more "inertial approach". Indeed the big difference between the EGI and GPS modules is that, while the former knows only its initial conditions and it has to compute the position from velocity, the latter works with the position already computed.

4.3.1.4.2 simscreen.cpp

```
1  void SimScreen::send_data(){
2
3      std::string c1_bin = numericToBinary<float>((float)((lon) * (1.0f / (1 <<
       31))), 32);
4      std::string c2_bin = numericToBinary<float>((float)((lat) * (1.0f / (1 <<
       31))), 32);
5      std::string alt_bin = numericToBinary<float>(((float)(alt) * (1.0f / (1 <<
        3))), 20);
6      ...
7
8      memcpy(wr.payload, (c1_bin+c2_bin+h_bin+...).c_str(), PACKET_SIZE);
9      this->serial.Write(handle, FACE::IOSS::Serial::WAIT_FOREVER, &wr,
       PACKET_SIZE, return_code);
10
11 #endif
12 }
```

In this function, lat, lon and alt info are first divided by the LSB, converted into binary strings of required number of bits (32 for the lat and lon, 20 for the altitude) and then copied into the payload of the message to be sent over the serial. Finally, the write action is performed.

4.4 ICD and decimal-to-binary transformation

Here the code for the decimal to binary and vice versa conversion is presented. This data conversion is needed to represent the transfer across the serial: indeed in a real world scenario the sensor would work with binary data (defined in ICDs such as 3.2 or 3.1) and transmit them onto the serial in a binary format. Thus, the sensor application changes the format from binary into decimal to perform operations and data fusion: indeed, the binary data has a representation that in most cases doesn't match the length of a data type for number of bits - this was done to balance precision and efficiency -, so a conversion to decimal is needed to bring back data to standard data types. The choice of create this algorithm to transfer bytes instead of the classical serialization such as *write((void*) &buf)* is due to the need of using a precise amount of bits for every datum instead of primitive data type to reproduce a more embedded level scenario like the ones in the avionics world.

4.4.1 Numeric to Binary

This first function here presented realizes a representation of the data of the required length: first the data is converted into binary format with fixed exponent and mantissa length parts; then, according to the number of required bits (which of course must be greater than number of bits dedicated to sign and exponent parts), the rest of mantissa bits are set to zero and cut out. The final representation is a concatenation of bits, with the last part being a reduction of the mantissa to ensure the total number of bits matches the required number passed as a function argument.

```
template<typename T>
std::string numericToBinary(T num, int num_of_bits) {

    std::bitset<sizeof(T) * 8> bits(*reinterpret_cast<unsigned long long*>(&
    num));
    return bits.to_string().substr(0, num_of_bits);

}
```

4.4.2 Binary to Numeric

The next function operates in the exact opposite direction with respect of the previous one: it converts from binary to decimal. Starting from the binary representation as a bit string, a sequence of zeros is added in order to reach the length of the data in the required primitive type. For example, if a data represented on 20 bits must be reconverted into numeric represent by a *double* on a 64bit machine, 44 bits, all set to zeros, must be added to the sequence before the conversion takes place, in order to have a valid length. After that, data can be converted back to numeric by *memcpy*, copying into a memory typed-cell, while an implicit cast is performed.

```
template<typename T>
T binaryToNumeric(const std::string& binaryString, int num_of_bits) {
    static_assert(std::is_integral<T>::value || std::is_floating_point<T>::
    value,
```

```
4                   "Type T must be integer or floating point");
5
6       std::string str = binaryString+std::string((sizeof(T) * 8 - num_of_bits),'
        0');
7       std::bitset<sizeof(T) * 8> bits(str);
8       T value;
9       std::memcpy(&value, &bits, sizeof(T));
10      return value;
11
12  }
```

Chapter 5

Achieved Results

5.1 Introduction

The main result achieved by this thesis study is the realization of a fully working implementation realizing the FACE Technical Standard 3.1 software architecture. IMA and MOSA paradigms have been followed during the development of the software trying to achieve modularity, portability and interoperability as much as possible.

5.2 Limitations

5.2.1 Documentation

The first encountered problem has been the lack of a code development documentation provided inside the technical standard. It is important to notice that the only complete example provided, which has revealed very much useful in writing the code and organizing it, was the IOSS Serial management. Also in this case, the code was superficially discussed and only methods signature were provided with little explanation but for the most cases it was sufficient for the practical code writing. No code explanation or example is provided for the several Capabilites mentioned in the standard: because of this, no Capability has been implemented in this current version, but in future versions, this aspect will be completely reconsidered. Thus, this code aims to be as compliant as possible to FACE Technical Standard [1], given all the previous limitations and difficulties.

5.2.2 Programming software

Another obstacle to the total FACE compliance is the absence of a proper HMFM system inside the OSS since for this a specific Real Time Operating System would be required. This type of software is not open source and it is extremely expensive to purchase. A RTOS-like OS would be preferable in this case to manage in a better way the different tasks of the applications. The use of a standard computer operating system with no knowledge of periodic or aperiodic tasks, limited the potential of the software. The greater limitation encountered in the code development is the

unavailability of the professional FACE software tools such as Skayl CINC & Phenom [20] or Ansys SCADE Architect [21]: these tools allow to write software in the different FACE Segments helping the programmer in the development and integration. The use of these tools is under evaluation for the future version of this software.

5.2.3 OSS

As already treated in the past chapters one of the major limitation in the FACE conformance and full potential is represented by the use of a Personal Computer Operating System instead of a real Time Operating System RTOS. Indeed, in nominal conditions, a personal PC with a *commercial* OS is not suitable unless an intervention is made with the use of other operating systems on the same machine. The *commercial* OS is thought for daily use on laptops and consumer devices while the RTOS is designed for real time systems with demanding requirements in terms of real time response, safety and task isolation/separation. Furthermore, an aspect of paramount importance is that RT Operating Systems allow to manually schedule tasks acting on the scheduler in order to achieve a deterministic scheduling while this is not possible on the standard PC OS since no concept of priority is related to processes or threads.

5.3 Hardware simplification and data representation

One simplification, impacting not the code organization but data processing, is caused by the use of a handmade flight simulator which neglects all realistic and complex dynamics. Another advantage, beside the simplicity of the algorithm, is the communication between sensors simulators and the rest of the architecture. The aspect that sensors are simulated simplifies the interaction since no driver must be used and no hardware complexity such as wiring, voltages and so on, must be known. Despite all these simplifications, the data representation aims to be as much realistic as possible being designed according to real ICDs. This makes the developed software employable in a real scenario with very little changes or adaptations.

5.4 Results

The primary outcome of this thesis study is the development of a fully functional software stack that adheres as closely as possible to the FACE Technical Standard 3.1, within the constraints previously discussed. Each layer has been investigated and analysed in a very precise manner in order to understand the provided potential and imposed restriction. This thesis work aims at providing an extremely simplified working example discussing the advantages and disadvantages brought in by the adoption of the FACE Architecture.

5.4.1 IMA, OSA and MOSA Principles in the proposed architecture

The software architecture here proposed, succeeded in realizing key principles of Modular Open Systems Approach (MOSA), Open Systems Architecture (OSA), and Integrated Modular Avionics (IMA):

- Modularity: The separation of applications (e.g., A/C Position app inside a segment and EGI, GPS, and Display Apps inside another one) and system components (such as I/O, timing services, and OS) highlights a modular structure. Each application and component can be updated or replaced independently, allowing scalability and easier upgrades.
- Interoperability: The OpenDDS middleware establishes an open communication infrastructure, promoting interoperability between modules across PSSS and PCS. This use of open standards aligns with MOSA, making integration with third-party components straightforward.
- Hardware Independence: The layered structure, from applications down to the OS and middleware, abstracts the underlying hardware, embodying IMA principles. This separation allows software modules - which are agnostic of low level specifications - to be transferred to different hardware platforms with minimal modification.
- Standardized Communication and IO APIs: The use of a standardized APIs enables communication across modules and with peripherals from different vendors, a core MOSA and OSA concept. This design fosters reusability and vendor competition.

5.4.2 Dynamic Library Implementation

Inside the Technical Standard the IOSS is defined as a service and as a library (IO Lib) for the PSSS entities, since no active component is contained inside but only passive ones to be included and used inside other software. In other words, no *main* file appears and no executable is produced. Thanks to this fact, the IOSS can be converted into a library, a static or dynamic one, to be included inside the *CMakeLists.txt* file of other software components. In addition, its functions can be called without listing all the files inside the *CMakeLists.txt* file, as it would be if the IOSS was only a bunch of headers to be included to have access to IO methods. The main advantage is that, during the creation of an executable, the compiler and the linker have only to link the library and not including all the file constituting the library itself inside the executable binary code. Furthermore, if a change occurred inside the library no re-compilation is needed for the software component since the former remains compatible with the latter and the two parts are independent and not seen as a unique block. The last advantage is that the library is realized as a dynamic library, as `dylib` on MacOS and as `so` on Linux, it gets loaded into memory at runtime without being linked before and without occupying unnecessary memory if

is not called, improving in performances and efficiency. The realization of a dynamic library instead of the use of a set of C files can be seen as an optimization since this implies a reduction in the references number inside the *CMakeLists.txt* file, which now is shorter and more easily readable. Furthermore, this development choice allows for space and time saving. The IOSS library has been realized for both architecture (Section 4.2.4) in order to be compatible with both PSSS software components - which are explicitly `arm64` - and with the universal build of the graphical part of "AC_SIM" - which uses Qt `x86_64` set of widgets - and all the code (library included) must be of the same architecture. This means that all the software components benefit from this migration from files to library and the *CMakeLists.txt* files are now improved and simplified in their structure. As explained in the next subsection, the IOSS library has been realized also for Linux operating system, obtaining a `so` (Shared Object) file to be included instead of the `dylib` in the Ubuntu version of the code.

5.4.3 Operating System support

As mentioned in the previous paragraph, in order to achieve as much conformance as possible with FACE Technical Standard, the original thesis software, which has been developed on a MacOS system, has been ported to a Linux Ubuntu 24.04 virtual machine. The first aspect to be noted is that, by doing this, a multiplatform support has been realized and the very same software is now running on MacOS and Ubuntu. The porting from MacOS to Linux is very straight forward since both operating system are Unix like and POSIX compatible: file inclusion in header must be adapted to the specific operating system, executable and libraries must be re-compiled and linked but all the rest remains the same without any significant modification. However, between the two versions, the one on MacOS and the one on Ubuntu, some significant differences exist: the former use sockets for the communication between the PCS A/C Position component and the UoP to show the AC position – in this case residing inside the PCS – while, in the latter, TSS is used for the communication between PCS (A/C Position) UoP and PSGS application. This implies that in the MacOS version the graphical app resides inside the PCS – being a PCS UoP – while, in the Linux version, it lies inside the PSSS – being a PSGS UoC –: this means that this second version is more compliant with the FACE Technical Standard.

5.4.4 OpenDDS and Qt integration

One of the most remarkable results achieved in this work regards the integration between the FACE TSS implementation provided by OpenDDS and the Qt framework. Inside the documentation of OpenDDS only one small example - *iShapes* - of this integration is presented but is very confusing and lacks of any documentation. In addition the project is compiled through MPC 4.2.2.5 including sources, headers and even other files which in the folder of the project are present with another file name. Thus, the first step to fully achieve an integration was to compile the *iShape*

project by means of a `CMakeLists.txt` file - instead of the MPC - understanding in detail how every part works. This was only the first part of the job since the provided example project is using base OpenDDS, with Publisher/Subscriber explicit logic, while, for the development of the FACE software architecture proposal, the FACE TSS interface was needed. Next step has been the real integration of FACE TSS APIs inside a Qt window: this implies the use of a thread on which a message receiver runs in a continuous loop probing for messages and passing those messages to the rendering logic.

5.5 Simulation Analysis

As described before, data flows from one extreme to the other, from the input simulator of the aircraft to the window representing the cockpit with all the plane information. This flow is made of different steps in which several software components talk to each other and data must undergo several transformation. In the current work, data goes from the simulator to the cockpit window passing through four different running applications before being displayed: this is necessary to be compliant to FACE Standard since each application has its own task and must reside inside a specific layer (PSSS, TSS, PCS) in order to guarantee the portability and reusability goals, as IMA paradigm and FACE Technical Standard expect. Said that, the real time requirements are satisfied considering the total system, despite the different stages. This strict separation of modules which fosters IMA modularity, portability and interoperability comes at the price of communication overhead increased and stack traversal time growth but this is negligible with respect to the enormous advantages this new approach introduces. This time increase is due to the presence of different levels to pass through without going directly from the source to destination. But, on the other hand, the presence of different levels ensures proper data transformation and tasks separation.

5.5.1 Real Time considerations

One of the most problematic aspect at the beginning was the simulation of a sensor failure and the error reporting to the cockpit window in terms of real time requirements. The solution to this problem has been realized by merging two techniques which guarantee a very fast response in the magnitude of few tenths of seconds, while, before this approach, latency was up to several seconds.

The first part of solution is to simulate the failure of a sensor by having the sensor simulator module not communicating any more to the respective app inside the PSSS and flushing the serial buffer in order to remove packets and speed up the status change signaling: the aim of this technique is to trigger a time out expiration on the PSSS serial reading side. The approach change was in the PSSS apps: the EGI and GPS apps are realized as an infinite while loop in which a read operation from the serial is performed by means of the IOSS APIs. The change in the approach lays in

this part: in the first draft of the software, when a time out error was detected since no packet was read in the available time interval, the FACE::TIME_OUT error was propagated to the next level, which is the PCS A/C Position App, by sending no more messages via TSS. However, this is very little efficient and conceptually wrong since this behaviour would mean that also the PSSS app (EGI or GPS) is malfunctioning: this is not the case under investigation, as the scenario being simulated involves a sensor failure. The problem here was also the latency given by the fact it would have taken to much time before detecting a missing packet by the PCS A/C Position application.

The solution to this problem, the second part of the overall solution, is that, if a timeout error is obtained by the PSSS app, a different packet is sent via TSS to PCS: the timestamp of the message is set to zero to signal that the transferred data is not valid and an error in the sensor has occurred. On the PCS side, once the message is retrieved via TSS and the timestamp is checked, if the latter is not zero the packet is carrying valid info, otherwise an error is detected. In both cases, the message is sent to the cockpit simulator via sockets and the status of the sensors is communicated by setting an enum with a value among $BOTH$, EGI, GPS or $NONE$ to indicate whether a failure occurred and, if so, which sensor failed.

By this approach, the latency in the error displayed on the cockpit window, by turning off a LED representing the sensor, is reduced from about 9-10 seconds to 0.3-0.6 seconds in order to guarantee real time performances.

5.6 Graphical Components

5.6.1 Simulation App: AC_SIM

In the following subsection, the simulator window is reported under form of several screenshots at different instants of the simulation, in order to depict the behaviour of the graphical component in the most disparate scenarios.

The first image, Figure 5.1, depicts the initial configuration of the simulator window in which both speeds, accelerations and attitude angles are zero. Throttle and yoke input are zero, so the plane is standing still. For the simulation, as initial coordinates, the position of Turin, Italy, has been chosen: coordinates expressed in degree, minutes and seconds are then 45°04'45"N 7°40'34"E.

The second image, Figure 5.2, depicts the appearance of the window during a simulation: as it can be seen, with respect to the initial conditions, almost all the text boxes have changed their content; now speeds, accelerations and attitude angles are different from zeros thanks to acting on throttle and yoke.

The third screenshot, Figure 5.3, is quite similar to the first image but the latter differs from the former in the POSITION INPUT frame text boxes where new initial conditions for the simulation have been typed in. Pressing on the SET button these data will be used as initial conditions for the simulation and will appear inside the text boxes of both GPS and EGI sensors frames. In this case, the plane is standing

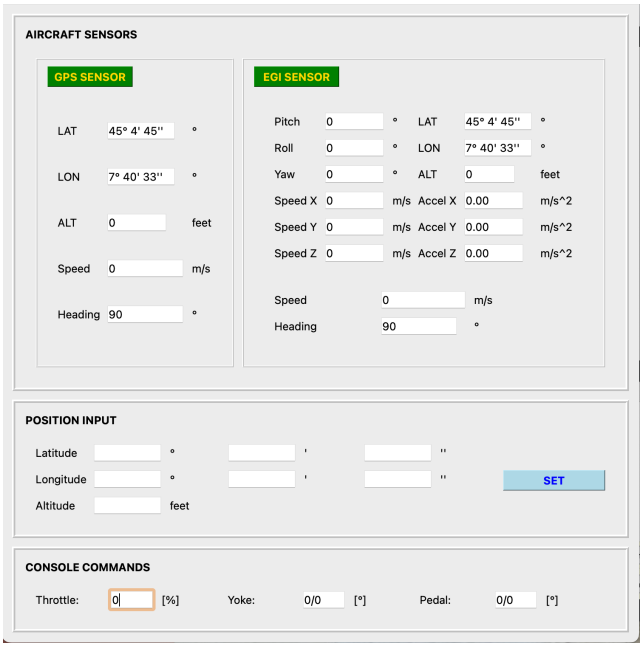

Figure 5.1: A/C SIM Window in initial configuration

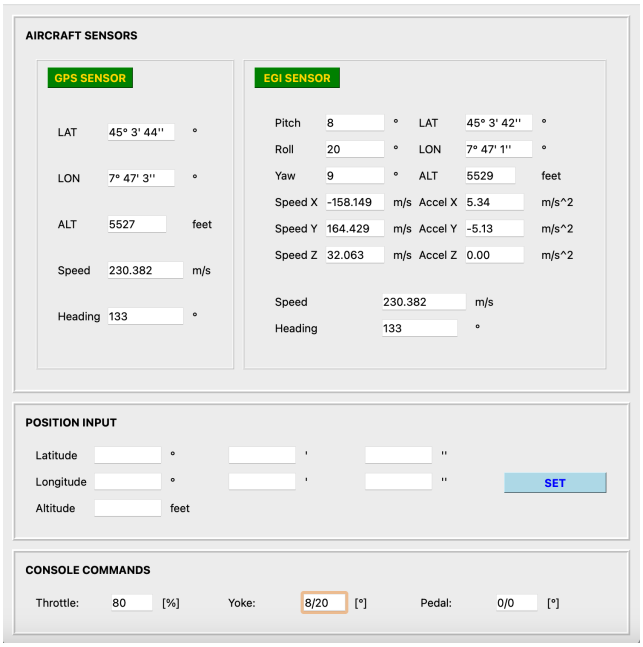

Figure 5.2: A/C SIM Window during a simulation

still to show the change in the coordinates without acting on speed and attitude but on the initial conditions only. Remind the difference in the functioning for the EGI and GPS modules, with the former being much more dependent to initial conditions with respect to the second. In the POSITION INPUT panel, several text boxes are present but it is no necessary for the user to fill all of them letting the possibility to act only on some specific data such as latitude only or altitude only.

The fourth image, Figure 5.4, depicts the effect of pressing the SET button to set new initial conditions, action that can be done at any moment during the simulation.

Figure 5.3: A/C SIM Window before setting new data

The reader is invited to compare Figure 5.3 and Figure 5.4 to check whether the same data have been transferred from the input boxes to the sensors text fields. Indeed, this new piece of info is used by the simulator to compute, as time passes, the evolution of the position of the plane. The very same initial conditions are passed to the EGI in order for it to compute the data since, as said before, this device has no notion of the "global info" but only of the initial condition and the current measurement which is used to compute the final data, in order to fake the inertial behaviour.

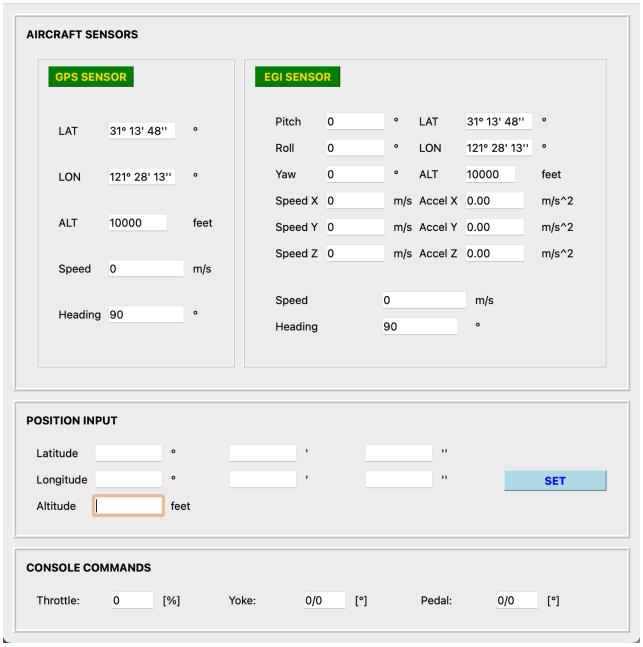

Figure 5.4: A/C SIM Window after setting new data

In Figure, 5.5, only the upper part of the simulator window is depicted: during the simulation, each panel (i.e. sensor) can be independently switched off in order to simulate a failure in the sensor and a missing transmission to the PSSS through serial. The panel keeps simulating the data but it doesn't transmit any more on the serial connection by means of IOSS. As explained before, in paragraph 5.5.1, the PSSS application for the specific sensor will experience a timeout error while reading and will communicate this malfunctioning to the A/C Position by a zero timestamp. This error will propagate through all the layers up to the Cockpit Simulator and will appear on the display.

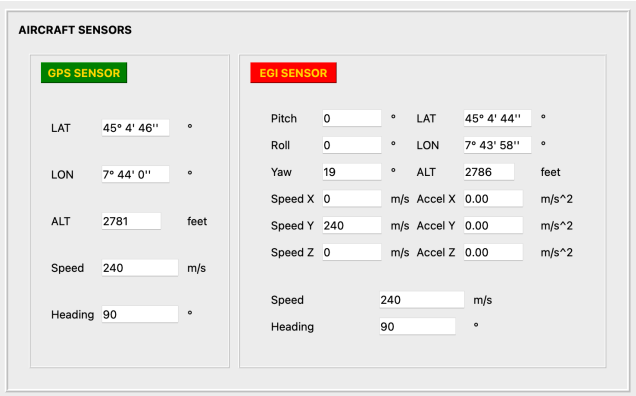

Figure 5.5: A/C SIM EGI sensor switched off (Upper Part only)

5.6.2 PSGS UoC: AC Information Display

In the following subsection, the visual representation of the PSGS UoC, displaying aircraft data in real time, is reported under form of multiple screenshots during a simulation.

The Figure 5.6 shows the aspect of the PSGS UoC window, at start, before moving the plane by acting on the simulator window as discussed in Section 5.6.1. The simulation is already running but, since no throttle is applied, everything is standing still. In the upper part of the left half, the AIRCRAFT DATA panel finds place: it shows the data retrieved by TSS from the A/C Position App. In the upper grid, information about position, speed and acceleration are showed, while, in the centre of the panel, there are heading and aircraft speed labels. In the bottom of the AC DATA panel, there are two labels and three buttons organized in a grid fashion. The EGI SENS and the GPS SENS labels work as LEDs to indicate to the user whether the sensors are working or not, that is, whether in the AC SIM window, the two simulator modules for EGI and GPS are enabled or disabled. The buttons, instead, are intended to be both LEDs and signal senders: the colour represents the status of the relative software applications (whether they are correctly running or they are stuck and failing). When they are green, apps are correctly running and confirmation data are received over TSS or IOSS; if they turn to red, an application is not sending confirmation messages any more and a failure has happened. An action is needed to solve the problem and, by pressing the buttons, a kill signal is

sent to the PSCS HM which, in turn, kills the process related to the failing app and launches it again.

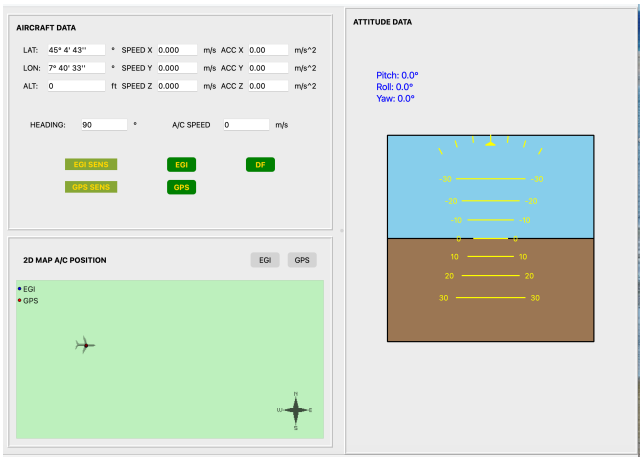

Figure 5.6: A/C POSITION APP in start configuration

In the second half of the left part, there is the 2D map for the A/C position. Near the panel name, two grey buttons allow the user to disable the depiction of one or both the position indicators of the plane: this is meant to let the user play around with the two dots and see the different positions retrieved by the two different sensors because of the measurement noise while the plane drawing represents the fused data by the A/C Position application. Indeed, as it is clearly visible in Figure 5.7, the plane is rolling on the right and is turning right while the heading is increasing (from 90° to 100°) in the AIRCRAFT DATA panel and the orientation of the drawing is rotating towards the bottom of the map around a z axis perpendicular to the screen plane.

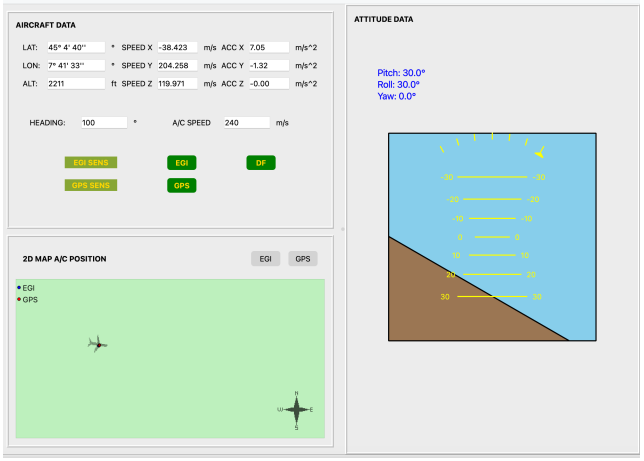

Figure 5.7: A/C POSITION APP during simulation of simultaneous pitch and roll manoeuvre

On the right half of the displayed window, inside the ATTITUDE DATA panel, there is a simplified version of the artificial horizon on which the pitch and roll angles are graphically displayed. As it can be clearly seen in Figure 5.7, pitch is *measured* through the yellow horizontal lines, while roll is *measured* by the dashed wheel and

the little triangle pointing towards the degree indicator on the wheel. Through complex geometry calculations, the extreme points of the horizon line can be found to depict polygons representing the ground and sky. The challenge lies in rendering the artificial horizon's division between ground and sky at extreme pitch and roll angles. When pitch is a small value and the plane is rolling, the horizon appears as an oblique line, forming two trapezoids. However, near extreme pitch values, one region (sky or ground) is represented as a triangle, while the other forms a pentagon, as illustrated in Figure 5.7.

Chapter 6

Conclusions

6.1 Introduction and Summary of Work

Military aviation systems are typically developed by single vendors for unique requirements, resulting in long lead times, costly upgrades, and limited hardware/software reuse across platforms. Rising complexity in mission equipment has further driven up costs and slowed new capability deployment. Procurement practices and the lack of standards complicate reuse, with additional challenges from the small market, aviation software qualification, and safety requirements beyond commercial standards. To address these issues, paradigms like IMA, MOSA, and OSA promote portability, reusability, and interoperability. In line with this shift, the FACE Technical Standard [1] accelerates software development, deployment, and integration in aviation to reduce time and costs.

The present study, conducted within the framework of this master's thesis, analyses, proposes and implements a software architecture according to the FACE Technical Standard model. Starting from the guidelines about code, provided inside the Technical Standard, different software components have been developed with the aim of casting light on strengths and weaknesses of FACE. The very focus which has driven all the software design and development has been the pursue for modularity of components and interoperability between applications.

6.2 Differences between FACE solution and non-FACE solution

This thesis work presents an implementation of a FACE software architecture designed to align closely with the Standard. However, several fundamental components are omitted, resulting in a streamlined solution that still adheres to the principles of FACE. An important analysis to be conducted is about the differences introduced by the adoption of the Standard with respect to an equal working non-FACE solution.

6.2.1 The importance of a FACE approach

The non-FACE solution is intended to be also a non IMA/MOSA solution which of course does not introduce the architectural changes of these new paradigms. In developing a non-FACE software solution, as opposed to the FACE-driven approach presented in this work, it's important to consider what elements should be omitted, which parts must remain unchanged, and which can be retained in a simplified form. This distinction helps clarify how the non-FACE solution can maintain essential functionalities while reducing complexity where possible, at the price of partially loosing modularity and interoperability. At first glance, code developed without adhering to FACE principles may seem simpler and have fewer lines of code, as the FACE architecture involves implementing various standards, layers of abstraction, and interfaces to ensure modularity, interoperability, and reusability. This often increases the software's complexity and line count. However, several aspects highlight the benefits of using the FACE approach:

- Modularity and Separation of Concerns → The FACE methodology encourages dividing the code into well-defined modules. Although this modularity can add files and lines of code, it enhances maintainability and lowers long-term complexity. Non-FACE-compliant code, while more compact, may lack modularity, making it harder to manage and update.
- Compatibility and Interoperability → FACE introduces interface layers and communication standards, facilitating integration with other systems. These layers may initially add to the code but help avoid compatibility issues that might later require substantial corrections or rewrites.
- Maintainability and Upgradability → Code not built on FACE principles may be shorter but is more likely to be monolithic and inflexible, making it difficult to adapt to new requirements. FACE's modular structure, in contrast, enhances software adaptability and simplifies the integration of new functionalities.

6.2.2 Non-FACE proposed solution in brief

In the following, a very brief analysis for each layer and software component is conducted.

6.2.2.1 AC Simulator

This component lays outside the FACE software architecture, so it is not interested by this non-FACE transformation. Of course, a change in the IOSS layer, would impact the way the simulator provides data to the EGI and GPS PSSS apps.

6.2.2.2 IOSS

In absence of this layer a direct access and use of POSIX APIs would be realized without a raster of methods whose aim is that of encapsulating those details. IO

communication would be managed by the single components, each of them using only the needed APIs, implementing different customized interfaces: this approach simplifies implementation, as no specific knowledge of IOSS library methods is required and, thus, case-specific solutions can be used. However, it also hinders reusability and modularity, as each component ends up with a unique way of interacting with the communication channels.

6.2.2.3 PSSS

EGI and GPS apps would communicate directly with the simulator modules without instantiating a Serial object to exchange data, using instead POSIX APIs (direct read/write operations on the serial descriptors), reducing again the modularity of the code. In a real context scenario, in the non-FACE solution, the usage of a serial connection would be managed by specific POSIX APIs without using high level functions. This would request a low level management of the connection since no encapsulation of low level details would be possible by IOSS libraries. In addition, with no separation between PSSS and PCS, the A/C Position UoP and the A/C Display UoC would be a unique application, implying less communication delay but, for certain, also less modularity without a specific software to compute the position and another specific software to show the aircraft data. Instead, the presence of two different components with different functions means that what one application does can be beneficial for other multiple other applications since it doesn't keep data inside itself but communicates with the outside.

6.2.2.4 TSS

This is one the most distinctive trait of the FACE standard which enhances interoperability and in a non-FACE solution this layer would be replaced by a direct connection between PSSS elements and PCS elements: this connection could be a serial, a shared file, a shared memory section or a socket. Of course, if no FACE layer is present, no division into PSSS and PCS would exist, since no concept of modularity or portability would be pursued. From a very technical point of view, the absence of this layer would probably remove the need for a transport mechanism, in this case DDS, reducing the code complexity. A viable solution to overcome the absence of the TS would be the usage of sockets, which are almost the same efficiency, simulating a publish-subscriber paradigm between client and server in a socket connection.

6.2.2.5 PCS

As for the other layers, the communication between A/C Position application and EGI and GPS apps or with the PSGS Display component would be realized through sockets or other Inter Process Communication methods. With respect to the current solution, the A/C Position app and the A/C data graphical widget (the PSGS UoC) could be part of the same application in which data are first received and then merged together without using the TSS since there would be no division in different segments

for these applications; this implies a very deep simplification of the structure and a more efficient data elaboration with no communication lag between the different parts, with the cost of no modularity and no tasks separation.

6.2.3 Segments latency coping techniques

The strict segmentation of software required by the FACE Standard introduces a longer latency with respect to the non-FACE case in which the communication is more straightforward and more direct without the participation of several layers. Some solutions can be adopted to cope with this phenomenon in order to minimize latency and optimise system response, without compromising modularity and interoperability:

- Use of high-performance middleware such as DDS, optimized for real-time data distribution, which reduces latency in communication between modules. Through QoS, it is possible define priorities, time requirements, update frequencies and packet sizes to optimize traffic and reduce overall latency.
- Reduction of intermediate steps, since, in some cases, it is possible directly connect two modules, without crossing intermediate levels when it is not necessary.
- Parallelism and Multi-threading by dividing workloads between different threads or CPU cores to execute concurrent processes, improving overall response time.
- Prioritization of operations by assigning a higher priority to critical processes, longer quantum time or more system resources.

6.2.4 The importance of the FACE Data Model

The FACE Shared Data Model (SDM) represents a key differentiator between FACE solutions and non-FACE solutions, primarily in terms of interoperability, data consistency, and ease of integration. The use of a standardized data model, where all the information handled by the system is clearly defined and documented, ensures that different applications can communicate with the assurance that they are exchanging the same data, characterized by the same types and meanings. In a non-FACE solution, where each application might use custom data formats, communication becomes much more challenging, as data structures need to be adapted or converted for each new integration. Furthermore, the integration of different software components, central to the concept of modularity, requires that modules be replaceable with minimal disruption to the rest of the system. This is achieved through common interfaces that allow the translation of data from one known format to another. Without a shared data model, replacing an application would likely necessitate changes to other elements of the architecture, which significantly complicates the process.

The FACE Data Model enables and enhances modularity and interoperability, offering several advantages:

- Interoperability → The SDM allows software modules developed by different suppliers to exchange data in a standardized manner, as it defines common data formats and shared structures. This ensures the seamless integration of components without the need for data adaptation or conversion, thereby reducing integration complexity.
- Data Consistency → By utilizing a centralized and shared data schema, the SDM ensures that all modules access and use data with the same structure and semantics, avoiding discrepancies or inconsistencies across the system.
- Component Reusability → With a common data standard, the SDM makes it possible to reuse software modules across different systems without significant modifications, promoting efficiency and reducing redundant work.
- Reduced Integration Complexity and Costs → Since modules use a common data model, there is no need to develop translators or adapters between different data formats. This simplifies the integration process and reduces the associated costs for testing and validating the system.
- Ease of Maintenance and Scalability → With a unified and shared data schema, it is easier to modify or update the system without having to adjust each component individually. This enhances the scalability of the architecture and makes it easier to expand and update over time.

In summary, the FACE Data Model facilitates an open, modular architecture where diverse components can collaborate seamlessly. This improves system compatibility and reduces the costs of integration and maintenance, making it a fundamental element for ensuring long-term system flexibility and efficiency. Without the SDM, these advantages would be significantly compromised, limiting modularity, interoperability, and the ability to scale and maintain the system effectively.

6.3 Cyber Security Considerations

One big challenge for the today society is security in the field of digital communication and computer applications: as technology advances also cyber threats evolve becoming ever more dangerous. In particular, in sensitive fields as military aviation, the concern for cyber security is of pivotal importance in order to cope with the current and future challenges, thus encrypted communication and data storage is necessary for this goal. In the context of this thesis, a very important improvement could be realized by creating a custom encryption protocol to achieve security of the treated information or to use some technologies for hardware or software encryption to prevent undesired access to sensitive data. As mentioned in Section 2.3.6, in IMA systems, information assurance is one of the main aspects of this paradigm and, since it is presented as realization of IMA/MOSA approaches, FACE provides a set of cybersecurity guidelines and practices to protect avionics systems and information from threats and

attacks. Since systems using FACE are modular and open, security is a crucial aspect to avoid vulnerabilities. FACE addresses cybersecurity by ensuring that each module complies with security requirements such as authentication, access control, encryption, and monitoring. Its modular architecture isolates components, minimizing the impact of vulnerabilities. The framework mandates that communications between modules be encrypted to prevent data interception and manipulation. To ensure that only authorized modules and users can access data or functionality, FACE incorporates authentication and access control mechanisms. Additionally, FACE promotes the use of monitoring and logging tools to track activities and detect any abnormal behaviours. The framework supports middleware like DDS, which provides integrated security features such as authentication and encryption. Furthermore, the modular design of FACE makes it easier to apply security updates and patches to individual modules without altering the entire system.

In the present work, however, no encryption has been implemented but, in a future version, this will be one of the first direction to move towards; furthermore, for this future implementation, remains to be established whether to realized encryption at a software level or at a hardware level to balance pros and cons and to achieve both efficiency and flexibility in communication mechanism.

6.4 Future work

Starting from what has been achieved in this current thesis study, possible developments and improvements can be realized in order to reach a even greater compliance with the FACE Technical Standard and a wider set of provided functionalities. Clearly, one possible direction of analysis is a very deep comprehension of the capabilities offered by the different FACE segments, understanding what this means in concrete terms and how it impacts the code and the algorithms. Another point of interest is the Shared Data Model, with the aim of realizing a very detailed data architecture able to represent a more realistic and a wider scenario than the one here discussed. This work would need the knowledge of data modelling software in order to realize a proper data model and the related code. On a different topic, the necessity of widening the set of supported communication protocols and mechanisms is evident. The IOSS (IO lib) should include other drivers used in avionics such as Ethernet, MIL-STD-1553, and specific drivers for avionics equipment, HW interfaces and sensors. As anticipated before, one priority concern should be the cybersecurity aspect related to every developed part of software, in particular those ones exchanging sensitive data. In this thesis, a cybersecurity overhaul should start from the TSS involving the PSSS-PCS communication and then a similar approach should be applied to the IOSS communication channels.

Appendix A

In this appendix, a very detailed copy of the source code is provided in order to complete the missing parts of the Chapter 4 and let the reader have a deep understanding of the code functioning in every part.

A.1 PCS

A.1.1 A/C Position Application

A.1.1.1 main.cpp

```
#include <iostream>

#include "FACE_LAYERS/TSS/TS/FaceMessage_TS.hpp"
#include "FACE_LAYERS/IOSS/FACE_IO.h"
#include "ace/Log_Msg.h"
#include "ace/OS_NS_unistd.h"
#include "OpenDDS-3.28.1/FACE/common.hpp"
#include "../DF_utils.h"
#include <chrono>
#include <thread>
#include <signal.h>

#ifdef ACE_AS_STATIC_LIBS
# include "dds/DCPS/RTPS/RtpsDiscovery.h"
# include "dds/DCPS/transport/rtps_udp/RtpsUdp.h"
#endif

#include "FACE_LAYERS/PSSS/AC_POS_APP_GRAPHIC/flag.h"

// Message to receive
Messenger::Position_EGI msg_TS_egi;
Messenger::Position_GPS msg_TS_gps;
Messenger::Monitor msg_TS_monitor;
Messenger::Position_DF msg_TS_df;

FACE::DF2ACP_message_t msg;

FACE::RETURN_CODE_TYPE status_egi;
FACE::RETURN_CODE_TYPE status_gps;
FACE::RETURN_CODE_TYPE status_monitor;
FACE::RETURN_CODE_TYPE status_df;


int msg_read[2] = {0,0};

FACE::CONNECTION_ID_TYPE connId_egi;
FACE::CONNECTION_ID_TYPE connId_mnt;
FACE::CONNECTION_ID_TYPE connId_gps;
FACE::CONNECTION_ID_TYPE connId_df;

void signal_handler(int signal, siginfo_t *info, void *extra){
    if(signal==SIGKILL){
```

```
43          printf("SIGKILL received!!!\n");
44          FACE::RETURN_CODE_TYPE status;
45
46  #ifdef USE_TSS
47          FACE::TS::Destroy_Connection(connId_df, status);
48          if(status!=FACE::NO_ERROR){
49              perror("DF=>Close connection error...");
50          }
51  #endif
52
53          FACE::TS::Destroy_Connection(connId_egi, status);
54          if(status!=FACE::NO_ERROR){
55              perror("DF=>Close connection error...");
56          }
57          FACE::TS::Destroy_Connection(connId_gps, status);
58          if(status!=FACE::NO_ERROR){
59              perror("DF=>Close connection error...");
60          }
61          FACE::TS::Destroy_Connection(connId_mnt, status);
62          if(status!=FACE::NO_ERROR){
63              perror("DF=>Close connection error...");
64          }
65          usleep(1000);
66      }
67      else{
68          perror("Ho detectato un altro segnale!");
69      }
70  }
71
72  int main() {
73
74      struct sigaction sa;
75      sa.sa_sigaction = signal_handler;
76      sa.sa_flags = SA_SIGINFO;
77      sigemptyset(&sa.sa_mask);
78      sigaction(SIGKILL, &sa, NULL);
79
80      FACE::RETURN_CODE_TYPE status;
81
82
83      printf("FACE::TS Initialization...\n");
84      FACE::TS::Initialize("/home/edocava/Desktop/Thesis/FACE_LAYERS/TSS/TS/
        config_static_DF.ini", status);
85
86      if (status != FACE::NO_ERROR) {
87          return static_cast<int>(status);
88      }
89
90      printf("FACE::TS Connection...\n");
91      // Create the sub_EGI connection
92
93      FACE::CONNECTION_DIRECTION_TYPE dir_egi;
94      FACE::MESSAGE_SIZE_TYPE max_msg_size_egi;
95      FACE::TS::Create_Connection(
96              "sub_EGI", FACE::PUB_SUB, connId_egi, dir_egi,
97              max_msg_size_egi, FACE::INF_TIME_VALUE, status);
98
99      if (status != FACE::NO_ERROR) {
100         perror("EGI...");
101         return static_cast<int>(status);
102     }
103
104     // Create the sub_GPS connection
105
106     FACE::CONNECTION_DIRECTION_TYPE dir_gps;
107     FACE::MESSAGE_SIZE_TYPE max_msg_size_gps;
108     FACE::TS::Create_Connection(
109             "sub_GPS", FACE::PUB_SUB, connId_gps, dir_gps,
110             max_msg_size_gps, FACE::INF_TIME_VALUE, status);
111
112     if (status != FACE::NO_ERROR) {
113         perror("GPS...");
114         return static_cast<int>(status);
115     }
116
```

```
117
118     // Create the pub_Monitor connection
119     FACE::CONNECTION_DIRECTION_TYPE dir_mnt;
120     FACE::MESSAGE_SIZE_TYPE max_msg_size_mnt;
121     FACE::TS::Create_Connection(
122             "pub_Monitor", FACE::PUB_SUB, connId_mnt, dir_mnt,
123             max_msg_size_mnt, FACE::INF_TIME_VALUE, status);
124
125     if (status != FACE::NO_ERROR) {
126         perror("DF=>Monitor...");
127         return static_cast<int>(status);
128     }
129
130 #ifdef USE_TSS
131     // Create the pub_DF connection
132     FACE::CONNECTION_DIRECTION_TYPE dir_df;
133     FACE::MESSAGE_SIZE_TYPE max_msg_size_df;
134     FACE::TS::Create_Connection(
135             "pub_DF", FACE::PUB_SUB, connId_df, dir_df,
136             max_msg_size_df, FACE::INF_TIME_VALUE, status);
137
138     if (status != FACE::NO_ERROR) {
139         perror("DF=>Monitor...");
140         return static_cast<int>(status);
141     }
142
143 #endif
144
145     FACE::TRANSACTION_ID_TYPE txn_egi;
146     FACE::TRANSACTION_ID_TYPE txn_gps;
147     FACE::TRANSACTION_ID_TYPE txn_mnt;
148     FACE::TRANSACTION_ID_TYPE txn_df;
149
150     printf("DF about to receive messages...\n");
151     fflush(stdout);
152
153
154     // open file only to remove content
155     int fd = open("/home/edocava/Desktop/Thesis/FACE_LAYERS/PCS/log.txt",
    O_WRONLY | O_TRUNC, 0644);
156     close(fd);
157
158     auto start = std::chrono::high_resolution_clock::now();
159
160     char buff[30];
161     while(1){
162
163         fd = open("/home/edocava/Desktop/Thesis/FACE_LAYERS/PCS/log.txt",
    O_WRONLY | O_APPEND, 777);
164         if(fd<=0){
165             printf("Permission denied!");
166             exit(0);
167         }
168
169         FACE::TS::Receive_Message(
170                 connId_egi, 1.5e8, txn_egi, msg_TS_egi,
171                 max_msg_size_egi, status_egi);
172
173
174
175         FACE::TS::Receive_Message(
176                 connId_gps, 1.5e8, txn_gps, msg_TS_gps,
177                 max_msg_size_gps, status_gps);
178
179
180
181
182
183         if (status_egi == FACE::NO_ERROR && status_gps == FACE::NO_ERROR) {
184
185             if(msg_TS_egi.timestamp>0 && msg_TS_gps.timestamp>0) { //BOTH
186
187                 memset((void*) &msg, 0, sizeof(FACE::DF2ACP_message_t));
188
189                 msg_TS_df.fuse_src = FACE::FUSION_TYPE::BOTH;
```

```
190                 msg_TS_df.timestamp = time(nullptr);
191
192                 printf("DF=>BOTH\n");
193                 fflush(stdout);
194
195                 /*
196                  * FUSED
197                  */
198
199                 msg_TS_df.speed = (msg_TS_egi.speed + msg_TS_gps.speed) / 2.0;
200
201                 msg_TS_df.att.pitch = msg_TS_egi.att.pitch;
202                 msg_TS_df.att.roll = msg_TS_egi.att.roll;
203                 msg_TS_df.att.yaw = msg_TS_egi.att.yaw;
204                 msg_TS_df.att.heading = (msg_TS_egi.att.heading + msg_TS_gps.
    att.heading) / 2.0;
205
206                 msg_TS_df.pos.lat = (msg_TS_egi.pos.lat + msg_TS_gps.pos.lat)
    / 2.0;
207                 msg_TS_df.pos.lon = (msg_TS_egi.pos.lon + msg_TS_gps.pos.lon)
    / 2.0;
208                 msg_TS_df.pos.alt = (msg_TS_egi.pos.alt + msg_TS_gps.pos.alt)
    / 2.0;
209
210                 msg_TS_df.speed_NED.X = msg_TS_egi.speed_NED.X;
211                 msg_TS_df.speed_NED.Y = msg_TS_egi.speed_NED.Y;
212                 msg_TS_df.speed_NED.Z = msg_TS_egi.speed_NED.Z;
213
214                 msg_TS_df.acc_NED.X = msg_TS_egi.acc_NED.X;
215                 msg_TS_df.acc_NED.Y = msg_TS_egi.acc_NED.Y;
216                 msg_TS_df.acc_NED.Z = msg_TS_egi.acc_NED.Z;
217
218                 /*
219                  * EGI
220                  */
221
222                 msg_TS_df.egi.speed = msg_TS_egi.speed;
223
224                 msg_TS_df.egi.pos.lat = msg_TS_egi.pos.lat;
225                 msg_TS_df.egi.pos.lon = msg_TS_egi.pos.lon;
226                 msg_TS_df.egi.pos.alt = msg_TS_egi.pos.alt;
227
228                 msg_TS_df.egi.att.heading = msg_TS_egi.att.heading;
229                 msg_TS_df.egi.att.pitch = msg_TS_egi.att.pitch;
230                 msg_TS_df.egi.att.roll = msg_TS_egi.att.roll;
231                 msg_TS_df.egi.att.yaw = msg_TS_egi.att.yaw;
232
233                 msg_TS_df.egi.speed_NED.X = msg_TS_egi.speed_NED.X;
234                 msg_TS_df.egi.speed_NED.Y = msg_TS_egi.speed_NED.Y;
235                 msg_TS_df.egi.speed_NED.Z = msg_TS_egi.speed_NED.Z;
236
237                 msg_TS_df.egi.acc_NED.X = msg_TS_egi.acc_NED.X;
238                 msg_TS_df.egi.acc_NED.Y = msg_TS_egi.acc_NED.Y;
239                 msg_TS_df.egi.acc_NED.Z = msg_TS_egi.acc_NED.Z;
240
241                 /*
242                  * GPS
243                  */
244
245                 msg_TS_df.gps.speed = msg_TS_gps.speed;
246                 msg_TS_df.gps.pos.lat = msg_TS_gps.pos.lat;
247                 msg_TS_df.gps.pos.lon = msg_TS_gps.pos.lon;
248                 msg_TS_df.gps.pos.alt = msg_TS_gps.pos.alt;
249                 msg_TS_df.gps.att.heading = msg_TS_gps.att.heading;
250
251
252
253 #ifdef USE_TSS
254                 FACE::TS::Send_Message(
255                     connId_df, FACE::INF_TIME_VALUE, txn_df, msg_TS_df,
256                     max_msg_size_df, status_df);
257                 if(status_df!=FACE::NO_ERROR){
258                     perror("DF=>Sending error...");
259                 }
260 #endif
```

```
261
262                 /*
263                     FILE LOG
264                 */
265                 msg.fusion = FACE::FUSION_TYPE::BOTH;
266                 msg.tmp = msg_TS_df.timestamp;
267                 /*
268                  * FUSED
269                  */
270
271                 msg.ac_speed = (msg_TS_egi.speed + msg_TS_gps.speed) / 2.0;
272
273                 msg.att.pitch = msg_TS_egi.att.pitch;
274                 msg.att.roll = msg_TS_egi.att.roll;
275                 msg.att.yaw = msg_TS_egi.att.yaw;
276                 msg.att.hd = (msg_TS_egi.att.heading + msg_TS_gps.att.heading)
    / 2.0;
277
278                 msg.pos.lat = (msg_TS_egi.pos.lat + msg_TS_gps.pos.lat) / 2.0;
279                 msg.pos.lon = (msg_TS_egi.pos.lon + msg_TS_gps.pos.lon) / 2.0;
280                 msg.pos.alt = (msg_TS_egi.pos.alt + msg_TS_gps.pos.alt) / 2.0;
281
282                 msg.speed.X = msg_TS_egi.speed_NED.X;
283                 msg.speed.Y = msg_TS_egi.speed_NED.Y;
284                 msg.speed.Z = msg_TS_egi.speed_NED.Z;
285
286                 msg.acc.X = msg_TS_egi.acc_NED.X;
287                 msg.acc.Y = msg_TS_egi.acc_NED.Y;
288                 msg.acc.Z = msg_TS_egi.acc_NED.Z;
289
290                 /*
291                  * EGI
292                  */
293
294                 msg.egi.ac_speed = msg_TS_egi.speed;
295
296                 msg.egi.pos.lat = msg_TS_egi.pos.lat;
297                 msg.egi.pos.lon = msg_TS_egi.pos.lon;
298                 msg.egi.pos.alt = msg_TS_egi.pos.alt;
299
300                 msg.egi.att.hd = msg_TS_egi.att.heading;
301                 msg.egi.att.pitch = msg_TS_egi.att.pitch;
302                 msg.egi.att.roll = msg_TS_egi.att.roll;
303                 msg.egi.att.yaw = msg_TS_egi.att.yaw;
304
305                 msg.egi.speed.X = msg_TS_egi.speed_NED.X;
306                 msg.egi.speed.Y = msg_TS_egi.speed_NED.Y;
307                 msg.egi.speed.Z = msg_TS_egi.speed_NED.Z;
308
309                 msg.egi.acc.X = msg_TS_egi.acc_NED.X;
310                 msg.egi.acc.Y = msg_TS_egi.acc_NED.Y;
311                 msg.egi.acc.Z = msg_TS_egi.acc_NED.Z;
312
313                 /*
314                  * GPS
315                  */
316
317                 msg.gps.speed = msg_TS_gps.speed;
318                 msg.gps.pos.lat = msg_TS_gps.pos.lat;
319                 msg.gps.pos.lon = msg_TS_gps.pos.lon;
320                 msg.gps.pos.alt = msg_TS_gps.pos.alt;
321                 msg.gps.att.hd = msg_TS_gps.att.heading;
322
323
324                 //printf("Message written...\n");
325                 write(fd, (void *) &msg, sizeof(FACE::DF2ACP_message_t));
326                 fsync(fd);
327
328             }
329             else if(msg_TS_egi.timestamp>0 && msg_TS_gps.timestamp==0){  //EGI
330
331                 memset((void*) &msg, 0, sizeof(FACE::DF2ACP_message_t));
332
333                 msg_TS_df.fuse_src = FACE::FUSION_TYPE::EGI;
334                 msg_TS_df.timestamp = time(nullptr);
```

```
335
336                 printf("DF=>EGI\n");
337                 fflush(stdout);
338
339                 msg_TS_df.speed = msg_TS_egi.speed;
340
341                 msg_TS_df.att.pitch = msg_TS_egi.att.pitch;
342                 msg_TS_df.att.roll = msg_TS_egi.att.roll;
343                 msg_TS_df.att.yaw = msg_TS_egi.att.yaw;
344                 msg_TS_df.att.heading = msg_TS_egi.att.heading;
345
346                 msg_TS_df.pos.lat = msg_TS_egi.pos.lat;
347                 msg_TS_df.pos.lon = msg_TS_egi.pos.lon;
348                 msg_TS_df.pos.alt = msg_TS_egi.pos.alt;
349
350                 msg_TS_df.speed_NED.X = msg_TS_egi.speed_NED.X;
351                 msg_TS_df.speed_NED.Y = msg_TS_egi.speed_NED.Y;
352                 msg_TS_df.speed_NED.Z = msg_TS_egi.speed_NED.Z;
353
354                 msg_TS_df.acc_NED.X = msg_TS_egi.acc_NED.X;
355                 msg_TS_df.acc_NED.Y = msg_TS_egi.acc_NED.Y;
356                 msg_TS_df.acc_NED.Z = msg_TS_egi.acc_NED.Z;
357
358
359                 //EGI ONLY
360                 msg_TS_df.egi.speed = msg_TS_egi.speed;
361
362                 msg_TS_df.egi.pos.lat = msg_TS_egi.pos.lat;
363                 msg_TS_df.egi.pos.lon = msg_TS_egi.pos.lon;
364                 msg_TS_df.egi.pos.alt = msg_TS_egi.pos.alt;
365
366                 msg_TS_df.egi.att.heading = msg_TS_egi.att.heading;
367                 msg_TS_df.egi.att.pitch = msg_TS_egi.att.pitch;
368                 msg_TS_df.egi.att.roll = msg_TS_egi.att.roll;
369                 msg_TS_df.egi.att.yaw = msg_TS_egi.att.yaw;
370
371                 msg_TS_df.egi.speed_NED.X = msg_TS_egi.speed_NED.X;
372                 msg_TS_df.egi.speed_NED.Y = msg_TS_egi.speed_NED.Y;
373                 msg_TS_df.egi.speed_NED.Z = msg_TS_egi.speed_NED.Z;
374
375                 msg_TS_df.egi.acc_NED.X = msg_TS_egi.acc_NED.X;
376                 msg_TS_df.egi.acc_NED.Y = msg_TS_egi.acc_NED.Y;
377                 msg_TS_df.egi.acc_NED.Z = msg_TS_egi.acc_NED.Z;
378
379
380 #ifdef USE_TSS
381                 FACE::TS::Send_Message(
382                     connId_df, FACE::INF_TIME_VALUE, txn_df, msg_TS_df,
383                     max_msg_size_df, status_df);
384                 if(status_df!=FACE::NO_ERROR){
385                     perror("DF=>Sending error...");
386                 }
387 #endif
388
389                 /*
390                     FILE LOG
391                 */
392                 msg.fusion = FACE::FUSION_TYPE::EGI;
393                 msg.tmp = msg_TS_df.timestamp;
394
395                 /*
396                  * EGI ONLY
397                  */
398
399                 msg.egi.ac_speed = msg_TS_egi.speed;
400
401                 msg.egi.pos.lat = msg_TS_egi.pos.lat;
402                 msg.egi.pos.lon = msg_TS_egi.pos.lon;
403                 msg.egi.pos.alt = msg_TS_egi.pos.alt;
404
405                 msg.egi.att.hd = msg_TS_egi.att.heading;
406                 msg.egi.att.pitch = msg_TS_egi.att.pitch;
407                 msg.egi.att.roll = msg_TS_egi.att.roll;
408                 msg.egi.att.yaw = msg_TS_egi.att.yaw;
409
```

```
410                 msg.egi.speed.X = msg_TS_egi.speed_NED.X;
411                 msg.egi.speed.Y = msg_TS_egi.speed_NED.Y;
412                 msg.egi.speed.Z = msg_TS_egi.speed_NED.Z;
413
414                 msg.egi.acc.X = msg_TS_egi.acc_NED.X;
415                 msg.egi.acc.Y = msg_TS_egi.acc_NED.Y;
416                 msg.egi.acc.Z = msg_TS_egi.acc_NED.Z;
417
418                 //printf("Message written...\n");
419                 write(fd, (void*) &msg, sizeof(FACE::DF2ACP_message_t));
420                 fsync(fd);
421
422
423             }
424             else if(msg_TS_egi.timestamp==0 && msg_TS_gps.timestamp>0){ //GPS
425
426                 memset((void*) &msg, 0, sizeof(FACE::DF2ACP_message_t));
427
428                 msg_TS_df.fuse_src = FACE::FUSION_TYPE::GPS;
429                 msg_TS_df.timestamp = time(nullptr);
430
431                 printf("DF=>GPS\n");
432                 fflush(stdout);
433
434                 msg_TS_df.speed = msg_TS_gps.speed;
435                 msg_TS_df.att.heading = msg_TS_gps.att.heading;
436                 msg_TS_df.pos.lat = msg_TS_gps.pos.lat;
437                 msg_TS_df.pos.lon = msg_TS_gps.pos.lon;
438                 msg_TS_df.pos.alt = msg_TS_gps.pos.alt;
439
440                 //GPS ONLY
441                 msg_TS_df.gps.speed = msg_TS_gps.speed;
442                 msg_TS_df.gps.att.heading = msg_TS_gps.att.heading;
443                 msg_TS_df.gps.pos.lat = msg_TS_gps.pos.lat;
444                 msg_TS_df.gps.pos.lon = msg_TS_gps.pos.lon;
445                 msg_TS_df.gps.pos.alt = msg_TS_gps.pos.alt;
446
447
448 #ifdef USE_TSS
449                 FACE::TS::Send_Message(
450                     connId_df, FACE::INF_TIME_VALUE, txn_df, msg_TS_df,
451                     max_msg_size_df, status_df);
452                 if(status_df!=FACE::NO_ERROR){
453                     perror("DF=>Sending error...");
454                 }
455 #endif
456
457                 /*
458                     FILE LOG
459                 */
460
461                 msg.fusion = FACE::FUSION_TYPE::GPS;
462                 msg.tmp = msg_TS_df.timestamp;
463
464                 /*
465                  * GPS ONLY
466                  */
467
468                 msg.gps.speed = msg_TS_gps.speed;
469                 msg.gps.pos.lat = msg_TS_gps.pos.lat;
470                 msg.gps.pos.lon = msg_TS_gps.pos.lon;
471                 msg.gps.pos.alt = msg_TS_gps.pos.alt;
472                 msg.gps.att.hd = msg_TS_gps.att.heading;
473
474                 //printf("Message written...\n");
475                 write(fd, (void*) &msg, sizeof(FACE::DF2ACP_message_t));
476                 fsync(fd);
477
478             }
479             else{
480                 // send the same old message
481
482                 msg_TS_df.fuse_src = FACE::FUSION_TYPE::NONE;
483                 msg_TS_df.timestamp = time(nullptr);
484
```

```
485                 msg.fusion = FACE::FUSION_TYPE::NONE;
486                 msg.tmp = msg_TS_df.timestamp;
487
488 #ifdef USE_TSS
489                 FACE::TS::Send_Message(
490                     connId_df, FACE::INF_TIME_VALUE, txn_df, msg_TS_df,
491                     max_msg_size_df, status_df);
492                 if(status_df!=FACE::NO_ERROR){
493                     perror("DF=>Sending error...");
494                 }
495 #endif
496
497                 write(fd, (void*) &msg, sizeof(FACE::DF2ACP_message_t));
498                 fsync(fd);
499
500                 perror("DF=>NO valid message received!!\n");
501             }
502         }
503
504         /*
505          * CASE EGI
506          */
507         else if (status_egi == FACE::NO_ERROR && status_gps != FACE::NO_ERROR)
     {
508             perror("EGI ONLY -- GPS reception error!");
509         }
510
511         /*
512          * CASE GPS
513          */
514         else if (status_egi != FACE::NO_ERROR && status_gps == FACE::NO_ERROR)
     {
515             perror("GPS ONLY -- EGI reception error!");
516         }
517
518         /*
519          * NO reception at all: both timeouts expired
520          */
521         else{
522             perror("No reception at all!");
523         }
524
525         close(fd);
526
527         auto end = std::chrono::high_resolution_clock::now();
528         auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(
    end - start).count();
529         if(duration>1e3){
530
531             msg_TS_monitor.timestamp = time(nullptr);
532             msg_TS_monitor.code = FACE::NO_ERROR;
533
534
535             FACE::TS::Send_Message(
536                     connId_mnt, FACE::INF_TIME_VALUE, txn_mnt, msg_TS_monitor,
537                     max_msg_size_mnt, status_monitor);
538             if(status_monitor!=FACE::NO_ERROR){
539                 perror("DF=>Sending error (to HM)...");
540             }
541
542             start = std::chrono::high_resolution_clock::now();
543         }
544
545     }
546
547 #ifdef USE_TSS
548     FACE::TS::Destroy_Connection(connId_df, status_df);
549     if(status_df!=FACE::NO_ERROR){
550         perror("DF=>Close connection error...");
551     }
552 #endif
553
554
555     return 0;
556 }
```

A.2 TSS

NOTE: for this section only IDL and configuration files are reported since all the other software is automatically generated and very difficult to read and understand.

A.2.1 FaceMessage.idl

```
/*
 * Distributed under the OpenDDS License.
 * See: http://www.opendds.org/license.html
 */

module Messenger {

  struct pos_t{
    double lat;
    double lon;
    long alt;
  };

  struct att_t{
    short pitch;
    short roll;
    short yaw;
    float heading;
  };

  struct acc_t{
    double X;
    double Y;
    double Z;
  };

  struct speed_t{
    double X;
    double Y;
    double Z;
  };


  @topic
  struct Position_EGI{
    pos_t pos;
    att_t att;
    double speed;
    speed_t speed_NED;
    acc_t acc_NED;
    @key long long timestamp;
  };


  @topic
  struct Position_GPS{
    pos_t pos;
    att_t att;  /* solo heading attivo */
    double speed;
    @key long long timestamp;
  };

  @topic
  struct Monitor{
    long code;
    @key long long timestamp;
  };

```

```
};
```

A.2.2 config_static.ini

```
[common]
DCPSGlobalTransportConfig=$file

# domains
[domain/3]
DiscoveryConfig=DEFAULT_STATIC

[domain/6]
DiscoveryConfig=DEFAULT_STATIC

[domain/9]
DiscoveryConfig=DEFAULT_STATIC

[rtps_discovery/uni_rtps]
SedpMulticast=0
ResendPeriod=2

# transport config
#PUB EGI
[transport/rtps_pub1]
transport_type=rtps_udp
use_multicast=0
local_address=127.0.0.1:21074

[config/pub_EGI]
transports=rtps_pub1

# SUB EGI
[transport/rtps_sub1]
transport_type=rtps_udp
use_multicast=0
local_address=127.0.0.1:21075

[config/sub_EGI]
transports=rtps_sub1

# PUB GPS
[transport/rtps_pub2]
transport_type=rtps_udp
use_multicast=0
local_address=127.0.0.1:21076

[config/pub_GPS]
transports=rtps_pub2

# SUB GPS
[transport/rtps_sub2]
transport_type=rtps_udp
use_multicast=0
local_address=127.0.0.1:21077

[config/sub_GPS]
transports=rtps_sub2

# PUB MONITOR
[transport/rtps_pub3]
transport_type=rtps_udp
use_multicast=0
local_address=127.0.0.1:21078

[config/pub_Monitor]
transports=rtps_pub3

# SUB MONITOR
[transport/rtps_sub3]
transport_type=rtps_udp
use_multicast=0
```

```
local_address=127.0.0.1:21079

[config/sub_Monitor]
transports=rtps_sub3

# topic
[topic/Position_EGI]
platform_view_guid=104
type_name=Messenger::Position_EGI
max_message_size=128

[topic/Position_GPS]
platform_view_guid=104
type_name=Messenger::Position_GPS
max_message_size=64

[topic/Monitor]
platform_view_guid=104
type_name=Messenger::Monitor
max_message_size=16


[connection/pub_EGI]
id=1
participantid=111111
domain=3
direction=source
topic=Position_EGI
datawriterqos=durable_writer
config=pub_EGI

[connection/sub_EGI]
id=2
participantid=111112
domain=3
direction=destination
topic=Position_EGI
datareaderqos=durable_reader
config=sub_EGI

[connection/pub_GPS]
id=3
participantid=111113
domain=6
direction=source
topic=Position_GPS
datawriterqos=durable_writer
config=pub_GPS

[connection/sub_GPS]
id=4
participantid=111114
domain=6
direction=destination
topic=Position_GPS
datareaderqos=durable_reader
config=sub_GPS

[connection/pub_Monitor]
id=5
participantid=111115
domain=9
direction=source
topic=Monitor
datawriterqos=durable_writer
config=pub_Monitor

[connection/sub_Monitor]
id=6
participantid=111116
domain=9
direction=destination
topic=Monitor
datawriterqos=durable_reader
config=sub_Monitor
```

```
143
144 [datawriterqos/durable_writer]
145 durability.kind=TRANSIENT_LOCAL
146 reliability.kind=RELIABLE
147
148 [datareaderqos/durable_reader]
149 durability.kind=TRANSIENT_LOCAL
150 reliability.kind=RELIABLE
```

A.3 PSSS

A.3.1 Utility

A.3.1.1 sim_utils.h

```
1  #ifndef SIM_UTILS_H
2  #define SIM_UTILS_H
3
4  #include <iostream>
5  #include <bitset>
6  #include <cstring> // For memcpy
7  #include <limits>  // For numeric_limits
8  #include <type_traits> // For is_floating_point, is_integral
9  #include "math.h"
10
11 const double PI = 3.14159265358979323846;
12
13 /* earth radius */
14 const double R = 6371000.0;
15
16 typedef struct speed_s{
17     double X;
18     double Y;
19     double Z;
20 }speed_t;
21
22 typedef struct pos_s{
23     double X;
24     double Y;
25     double Z;
26 }pos_t;
27
28 typedef struct acc_s{
29     double X;
30     double Y;
31     double Z;
32 }acc_t;
33
34 const double EARTH_RADIUS = 6378137.0;
35 const double DEG_TO_RAD = M_PI / 180.0;
36 const double RAD_TO_DEG = 180.0 / M_PI;
37
38 typedef struct Coordinate_s{
39     int degree;
40     int min;
41     double sec;
42 }Coordinate_t;
43
44 template<typename T>
45 std::string numericToBinary(T num, int num_of_bits) {
46
47     std::bitset<sizeof(T) * 8> bits(*reinterpret_cast<unsigned long long*>(&
       num));
48     return bits.to_string().substr(0, num_of_bits);
49
50 }
51
52 template<typename T>
```

```
T binaryToNumeric(const std::string& binaryString, int num_of_bits) {
    static_assert(std::is_integral<T>::value || std::is_floating_point<T>::
    value,
                  "Type T must be integer or floating point");

    std::string str = binaryString+std::string((sizeof(T) * 8 - num_of_bits),'
    0');
    std::bitset<sizeof(T) * 8> bits(str);
    T value;
    std::memcpy(&value, &bits, sizeof(T));
    return value;

}

void compress(char *packet, const char *msg, int packet_len);
void uncompress(char *msg, char *packet, int packet_len);

double deg2rad(double angle);
double rad2deg(double angle);
void nedToLatLon(double *lat, double *lon, double dN, double dE);
void nedToLatLon2(double *lat0, double *lon0, double dN, double dE);
void decToDMS(const double decimalDegrees, Coordinate_t *c);
Coordinate_t decToDMS_r(const double decimalDegrees);
void dmsToDec(double *decimalDegrees, const Coordinate_t c);
double dmsToDec_r(const Coordinate_t c);


#endif // SIM_UTILS_H
```

A.3.1.2 sim_utils.cpp

```
//
// Created by Edoardo Cavallotti on 28/06/24.
//

#include "sim_utils.h"

double deg2rad(double angle){
    return angle/180.0*M_PI;
}

double rad2deg(double angle){
    return angle*180.0/M_PI;
}

/*
 * [dN] = m
 * [dE] = m
 */
void nedToLatLon(double *lat, double *lon, double dN, double dE) {

    double latRad = *lat * DEG_TO_RAD;
    double lonRad = *lon * DEG_TO_RAD;

    double dLatDegInMeters = 111320.0;
    double dLonDegInMeters = 111320.0 * cos(latRad);

    *lat += (dN / dLatDegInMeters) * RAD_TO_DEG;
    *lon += (dE / dLonDegInMeters) * RAD_TO_DEG;
}

void nedToLatLon2(double *lat0, double *lon0, double dN, double dE){

    double delta_lat, delta_lon;

    // Calculate the change in latitude
    delta_lat = dN / R * (180.0 / PI);

    // Calculate the change in longitude
    if (fabs(*lat0) < 90.0) { // Ensure the latitude is not at the poles
        delta_lon = dE / (R * cos(*lat0 * PI / 180.0)) * (180.0 / PI);
```

```
    } else {
        delta_lon = 0.0; // No change in longitude at the poles
    }

    *lat0 += delta_lat;
    *lon0 += delta_lon;
}

void decToDMS(const double decimalDegrees, Coordinate_t *c) {
    // Ottieni il valore assoluto dei gradi per calcolare i minuti e i secondi
    double absDecimalDegrees = fabs(decimalDegrees);

    // Calcola i gradi, minuti e secondi
    c->degree = (int) absDecimalDegrees;
    double decimalMinutes = (absDecimalDegrees - (double) c->degree) * 60.0;
    c->min = (int) decimalMinutes;
    c->sec = (decimalMinutes - (double) c->min) * 60.0;

    // Imposta il segno dei gradi a seconda del segno dell'input
    if (decimalDegrees < 0) {
        c->degree = -c->degree;
    }
}

Coordinate_t decToDMS_r(const double decimalDegrees) {
    Coordinate_t c;
    // Ottieni il valore assoluto dei gradi per calcolare i minuti e i secondi
    double absDecimalDegrees = fabs(decimalDegrees);

    // Calcola i gradi, minuti e secondi
    c.degree = (int) absDecimalDegrees;
    double decimalMinutes = (absDecimalDegrees - (double) c.degree) * 60.0;
    c.min = (int) decimalMinutes;
    c.sec = (decimalMinutes - (double) c.min) * 60.0;

    // Imposta il segno dei gradi a seconda del segno dell'input
    if (decimalDegrees < 0) {
        c.degree = -c.degree;
    }
    return c;
}

void dmsToDec(double *decimalDegrees, const Coordinate_t c){
    *decimalDegrees = (double)c.degree+(double)c.min/60.0+c.sec/3600.0;
}

double dmsToDec_r(const Coordinate_t c){
    return (double)c.degree+(double)c.min/60.0+c.sec/3600.0;
}

void compress(char *packet, const char *msg, int packet_len){
    for(int i=0; i<packet_len; i++){
        packet[i] = (char) 0;
        for(int j=0; j<8; j++){
            if(msg[i*8+j]==49){
                packet[i]+=1<<j;
            }
        }
    }
}

void uncompress(char *msg, char *packet, int packet_len){
    for(int i=0; i<packet_len; i++){
        for(int j=0; j<8; j++){
            msg[i*8+j] = '0';
            if(packet[i] & 1<<j){
                msg[i*8+j] = '1';
            }
        }
    }
}
```

A.3.2 EGI

A.3.2.1 EGI.h

```
//
// Created by Edoardo Cavallotti on 24/06/24.
//

#ifndef EGI_EGI_H
#define EGI_EGI_H

#include "FACE_LAYERS/IOSS/FACE_Serial/include/FACE_Serial.h"
#include "OpenDDS-3.28.1/FACE/common.hpp"
#include "../sim_utils.h"
#include "../DPM/DPM.h"
#include "../DPM/backup.h"
#include "AC_SIM/sim_constant.h"

class EGI {
public:
    void Initialize();
    void Open();
    void Read(messaggio_EGI_t *msg, FACE::RETURN_CODE_TYPE& return_status);
    void Close();
    void Flush();
private:
    FACE_Serial serial;
    FACE_Serial serialDPM;

    Coordinate_t lat;
    Coordinate_t lon;
    int height;

    speed_t speed_NED;
    acc_t acc_NED;

    float heading; //radians
    short pitch;
    short roll;
    short yaw;

    double speed;

    FACE::IOSS::Serial::CONNECTION_NAME_TYPE name;
    FACE::IOSS::CONNECTION_HANDLE_TYPE handle;

    FACE::IOSS::Serial::READ_PAYLOAD_TYPE rd;

    FACE::IOSS::Serial::CONNECTION_NAME_TYPE nameDPM;
    FACE::IOSS::CONNECTION_HANDLE_TYPE handleDPM;

#ifdef COMPRESS
    char *uncmp_packet;
#endif

};


#endif //EGI_EGI_H
```

A.3.2.2 EGI.cpp

```
//
// Created by Edoardo Cavallotti on 24/06/24.
//

#include "EGI.h"

void EGI::Initialize(){
```

```
8
9       lat.degree = 45;
10      lat.min = 4;
11      lat.sec = 45;
12
13      lon.degree = 7;
14      lon.min = 40;
15      lon.sec = 34;
16
17      height = 0;
18      speed = 0.0;
19
20      rd.payload = (FACE::IOSS::Serial::PAYLOAD_DATA_MSG_TYPE) malloc(sizeof(
        char)*EGI_PACKET_SIZE);
21
22      FACE::RETURN_CODE_TYPE return_code;
23      FACE::CONFIGURATION_RESOURCE config = "/Users/edoardocavallotti/Desktop/
        TESI/FACE_SOFTWARE_IMPLEMENTATION/FACE_LAYERS/PSSS/EGI/serialSIM";
24      serial.Initialize(config, return_code);
25      if(return_code!=FACE::NO_ERROR){
26          perror("EGI=>Error!!");
27      }
28
29      if(return_code!=FACE::NO_ERROR){
30          perror("EGI=>Error!!");
31      }
32
33  #ifdef COMPRESS
34      uncmp_packet = (char*) malloc(sizeof(char)*EGI_PACKET_SIZE*8);
35  #endif
36  }
37
38  void EGI::Open() {
39
40      FACE::RETURN_CODE_TYPE return_code;
41
42      serial.Open_Connection(name, FACE::IOSS::Serial::WAIT_FOREVER, handle,
        return_code);
43      if(return_code!=FACE::NO_ERROR){
44          perror("EGI=>Error!!");
45      }
46
47  }
48
49  void EGI::Read(messaggio_EGI_t *msg, FACE::RETURN_CODE_TYPE& return_status){
50
51      serial.Read(handle, FACE::IOSS::Serial::NO_WAIT, &rd, EGI_PACKET_SIZE,
        return_status);
52
53      switch(return_status){
54
55          case FACE::NO_ERROR:{
56
57  #ifdef COMPRESS
58
59              uncompress(uncmp_packet, rd.payload, EGI_PACKET_SIZE);
60
61              std::string str1 = string(uncmp_packet).substr(0,32);
62              std::string str2 = string(uncmp_packet).substr(32,32);
63              std::string str3 = string(uncmp_packet).substr(64,20);
64              std::string str4 = string(uncmp_packet).substr(84,16);
65
66              std::string str5 = string(uncmp_packet).substr(100,16);
67
68              std::string str6 = string(uncmp_packet).substr(116,16);
69              std::string str7 = string(uncmp_packet).substr(132,16);
70              std::string str8 = string(uncmp_packet).substr(148,16);
71
72              std::string str9 = string(uncmp_packet).substr(164,32);
73              std::string str10 = string(uncmp_packet).substr(196,32);
74              std::string str11 = string(uncmp_packet).substr(228,32);
75
76              std::string str12 = string(uncmp_packet).substr(260,16);
77              std::string str13 = string(uncmp_packet).substr(276,16);
78              std::string str14 = string(uncmp_packet).substr(292,16);
```

```
#else
            std::string str1 = string(rd.payload).substr(0,32);
            std::string str2 = string(rd.payload).substr(32,32);
            std::string str3 = string(rd.payload).substr(64,20);
            std::string str4 = string(rd.payload).substr(84,16);

            std::string str5 = string(rd.payload).substr(100,16);

            std::string str6 = string(rd.payload).substr(116,16);
            std::string str7 = string(rd.payload).substr(132,16);
            std::string str8 = string(rd.payload).substr(148,16);

            std::string str9 = string(rd.payload).substr(164,32);
            std::string str10 = string(rd.payload).substr(196,32);
            std::string str11 = string(rd.payload).substr(228,32);

            std::string str12 = string(rd.payload).substr(260,16);
            std::string str13 = string(rd.payload).substr(276,16);
            std::string str14 = string(rd.payload).substr(292,16);
#endif

            // ACCORDING TO ICD
            double c1 = (double) ((binaryToNumeric<float>(str1, 32)) * (1 << 31));
            double c2 = (double) ((binaryToNumeric<float>(str2, 32)) * (1 << 31));

            height = (int) ((binaryToNumeric<float>(str3, 20)) * (1 << 3));

            decToDMS(c1, &lon);
            decToDMS(c2, &lat);

            speed = (unsigned int) ((binaryToNumeric<float>(str4, 16)) * (1 << 0));

            heading = (float) ((binaryToNumeric<float>(str5, 16)) * (1 << 15)) ;  //in radians

            pitch = (short) ((binaryToNumeric<float>(str6, 16)) * (1 << 15));
            roll = (short) ((binaryToNumeric<float>(str7, 16)) * (1 << 15));
            yaw = (short) ((binaryToNumeric<float>(str8, 16)) * (1 << 15));

            speed_NED.X = (double) ((binaryToNumeric<float>(str9, 32)) * (1 << 18));
            speed_NED.Y = (double) ((binaryToNumeric<float>(str10, 32)) * (1 << 18));
            speed_NED.Z = (double) ((binaryToNumeric<float>(str11, 32)) * (1 << 18));

            acc_NED.X = (double) ((binaryToNumeric<float>(str12, 16)) * (1 << 5));
            acc_NED.Y = (double) ((binaryToNumeric<float>(str13, 16)) * (1 << 5));
            acc_NED.Z = (double) ((binaryToNumeric<float>(str14, 16)) * (1 << 5));


            msg->pos.alt = height;
            msg->pos.lon = c1;
            msg->pos.lat = c2;
            msg->att.hd = heading;
            msg->att.pitch = pitch;
            msg->att.roll = roll;
            msg->att.yaw = yaw;
            msg->speed.X = speed_NED.X;
            msg->speed.Y = speed_NED.Y;
            msg->speed.Z = speed_NED.Z;

            msg->acc.X = acc_NED.X;
            msg->acc.Y = acc_NED.Y;
            msg->acc.Z = acc_NED.Z;

            msg->ac_speed = speed;

            msg->tmp = rd.received_time;
```

```
144
145             break;
146         }
147         case FACE::NO_ACTION:
148             perror("EGI=>");
149             break;
150         case FACE::NOT_AVAILABLE:{
151             perror("EGI=>");
152             break;
153         }
154         case FACE::INVALID_PARAM:
155             perror("EGI=>");
156             break;
157         case FACE::INVALID_CONFIG:
158             perror("EGI=>");
159             break;
160         case FACE::INVALID_MODE:
161             perror("EGI=>");
162             break;
163         case FACE::TIMED_OUT:
164             perror("EGI=>Time out occurred!");
165             msg->tmp = 0; //flag to signal no packet has arrived
166             break;
167         case FACE::ADDR_IN_USE:
168             perror("EGI=>");
169             break;
170         case FACE::PERMISSION_DENIED:
171             perror("EGI=>");
172             break;
173         case FACE::MESSAGE_STALE:
174             perror("EGI=>");
175             break;
176         case FACE::IN_PROGRESS:
177             perror("EGI=>");
178             break;
179         case FACE::CONNECTION_CLOSED:
180             perror("EGI=>");
181             break;
182         case FACE::DATA_BUFFER_TOO_SMALL:
183             perror("EGI=>");
184             break;
185         case FACE::DATA_OVERFLOW:
186             perror("EGI=>");
187             break;
188     }
189
190
191 }
192
193 void EGI::Close() {
194     FACE::RETURN_CODE_TYPE return_code;
195     serial.Close_Connection(handle, return_code);
196 #ifdef COMPRESS
197     free(uncmp_packet);
198 #endif
199 }
200
201 void EGI::Flush() {
202     FACE::RETURN_CODE_TYPE return_code;
203     serial.Flush(handle, return_code);
204 }
```

A.3.2.3 EGI_manager.h

```
1 //
2 // Created by Edoardo Cavallotti on 02/11/24.
3 //
4
5 #ifndef PSSS_EGI_MANAGER_H
6 #define PSSS_EGI_MANAGER_H
7
```

```
#include <iostream>
#include "unistd.h"
#include "fcntl.h"
#include <cmath>
#include "EGI.h"

#include "FACE_LAYERS/TSS/TS/FaceMessage_TS.hpp"
#include "ace/Log_Msg.h"
#include "ace/OS_NS_unistd.h"

#include <chrono>
#include <thread>
#include <signal.h>

#define MONITOR_PACKET_SIZE 16

#ifdef ACE_AS_STATIC_LIBS
# include "dds/DCPS/RTPS/RtpsDiscovery.h"
# include "dds/DCPS/transport/rtps_udp/RtpsUdp.h"
#endif

typedef struct monitor_packet_s{
    long long timestamp;
    int code;
}monitor_packet_t;

#ifdef COMPRESS
char *uncmp_DPM_packet;
#endif

class EGI_manager{
public:

    static EGI_manager* getMyEGI();

    int Initialize();
    int Loop();
    int Close();
    int Destroy_TS_Connection();
    int Flush();
private:

    static EGI_manager *myEGI_man;
    EGI_manager();
    ~EGI_manager();

    EGI_manager(const EGI_manager&) = delete;
    EGI_manager& operator=(const EGI_manager&) = delete;

    EGI myEGI;

    FACE::IOSS::Serial::CONNECTION_NAME_TYPE serial_name;
    FACE::IOSS::Serial::WRITE_PAYLOAD_TYPE wr;

    FACE_Serial serial_monitor;
    FACE::IOSS::CONNECTION_HANDLE_TYPE serial_handle;
    FACE::CONNECTION_ID_TYPE connId;

    FACE::CONNECTION_DIRECTION_TYPE dir;
    FACE::MESSAGE_SIZE_TYPE max_msg_size;
    FACE::TRANSACTION_ID_TYPE txn;

    monitor_packet_t mnt_msg;

    // Message to send
    Messenger::Position_EGI msg_TS;

    long start;
    short to_cnt;

    messaggio_EGI_t msg;
    messaggio_EGI_t msg_old;

};

```

```
#endif //PSSS_EGI_MANAGER_H
```

A.3.2.4 EGI_manager.cpp

```
//
// Created by Edoardo Cavallotti on 02/11/24.
//

#include "EGI_manager.h"

EGI_manager* EGI_manager::myEGI_man = nullptr;

EGI_manager::EGI_manager(){}

EGI_manager::~EGI_manager(){}

EGI_manager* EGI_manager::getMyEGI() {
    if (myEGI_man == nullptr) {
        myEGI_man = new EGI_manager();
    }
    return myEGI_man;
}

int EGI_manager::Initialize(){
    FACE::RETURN_CODE_TYPE status;

    FACE::CONFIGURATION_RESOURCE config = "/Users/edoardocavallotti/Desktop/
    TESI/FACE_SOFTWARE_IMPLEMENTATION/FACE_LAYERS/PSSS/EGI/EGI_ser_monitor";
    serial_monitor.Initialize(config, status);
    serial_monitor.Open_Connection(serial_name, FACE::IOSS::Serial::
    WAIT_FOREVER, serial_handle, status);
    wr.payload = (FACE::IOSS::Serial::PAYLOAD_DATA_MSG_TYPE) malloc(sizeof(
    monitor_packet_t));

    FACE::TS::Initialize("/Users/edoardocavallotti/Desktop/TESI/
    FACE_SOFTWARE_IMPLEMENTATION/FACE_LAYERS/TSS/TS/config_static_EGI.ini",
    status);

    if (status != FACE::NO_ERROR) {
        return static_cast<int>(status);
    }

    myEGI.Initialize();
    myEGI.Open();

    // Create the pub connection
    FACE::TS::Create_Connection(
            "pub_EGI", FACE::PUB_SUB, connId, dir,
            max_msg_size, FACE::INF_TIME_VALUE, status);

    if (status != FACE::NO_ERROR) {
        return static_cast<int>(status);
    }

#ifdef COMPRESS
    uncmp_DPM_packet = (char*) malloc(sizeof(char)*EGI_PACKET_SIZE*8);
#endif

    // Send message

    ACE_DEBUG((LM_INFO, "Publisher: about to Send_Message()\n"));

    start = 0;
    to_cnt = 0;

}

int EGI_manager::Loop(){

    auto start_mnt = std::chrono::high_resolution_clock::now();
    FACE::RETURN_CODE_TYPE status;
```

```
63
64      while(1){
65          memset((void*) &msg, 0, sizeof(messaggio_EGI_t));
66          memset((void*) &mnt_msg, 0, sizeof(monitor_packet_t));
67          myEGI.Read(&msg, status);
68          switch(status){
69              case FACE::NO_ERROR:
70                  to_cnt = 0;
71                  msg_TS.speed = msg.ac_speed;
72
73                  msg_TS.speed_NED.X = msg.speed.X;
74                  msg_TS.speed_NED.Y = msg.speed.Y;
75                  msg_TS.speed_NED.Z = msg.speed.Z;
76
77                  msg_TS.pos.lon = msg.pos.lon;
78                  msg_TS.pos.lat = msg.pos.lat;
79                  msg_TS.pos.alt = msg.pos.alt;
80
81                  msg_TS.att.roll = msg.att.roll;
82                  msg_TS.att.pitch = msg.att.pitch;
83                  msg_TS.att.yaw = msg.att.yaw;
84                  msg_TS.att.heading = msg.att.hd;
85
86                  msg_TS.acc_NED.X = msg.acc.X;
87                  msg_TS.acc_NED.Y = msg.acc.Y;
88                  msg_TS.acc_NED.Z = msg.acc.Z;
89                  msg_TS.timestamp = msg.tmp;
90
91                  FACE::TS::Send_Message(
92                          connId, FACE::INF_TIME_VALUE, txn, msg_TS,
93                          max_msg_size, status);
94
95                  /*printf("EGI Sending Message... %ld\n", msg_TS.timestamp);
96                  fflush(stdout);*/
97
98                  if (status != FACE::NO_ERROR) {
99                      perror("EGI_m=>Impossible to send the message!");
100                     return static_cast<int>(status);
101                 }
102
103                 msg_old = msg;
104
105                 break;
106             case FACE::NO_ACTION:
107                 break;
108             case FACE::NOT_AVAILABLE:
109 #ifdef SERVER_BACKUP
110                 if(start>0){
111                 if(time(0)-start>10){
112                     perror("EGI=>Recover mode...\n");
113                     // prepare connection resources
114                     FACE::CONFIGURATION_RESOURCE config = "/Users/
    edoardocavallotti/Desktop/TESI/FACE_SOFTWARE_IMPLEMENTATION/FACE_LAYERS/
    PSSS/EGI/serialDPM";
115                     serialDPM.Initialize(config, return_code);
116                     if(return_code!=FACE::NO_ERROR){
117                         perror("EGI=>Error!!");
118                     }
119
120                     //open connection
121                     serialDPM.Open_Connection(nameDPM, FACE::IOSS::Serial::
    WAIT_FOREVER, handleDPM, return_code);
122                     if(return_code!=FACE::NO_ERROR){
123                         perror("EGI=>Error!!");
124                     }
125
126                     FACE::IOSS::Serial::WRITE_PAYLOAD_TYPE wr;
127                     wr.payload = (FACE::IOSS::Serial::PAYLOAD_DATA_MSG_TYPE)
    malloc(sizeof(char)*1);
128                     char ok = (char) 1;
129                     memcpy(wr.payload, (void*) &ok, sizeof(char));
130                     serialDPM.Write(handleDPM, FACE::IOSS::Serial::NO_WAIT, &
    wr, 1, return_code);  //to unlock the DPM service
131                     while(1){ // read data from DPM
```

```
132                         serialDPM.Read(handleDPM, FACE::IOSS::Serial::
    WAIT_FOREVER, &rd, EGI_PACKET_SIZE, return_code);
133                         if(return_code==FACE::NO_ERROR) {
134                             std::string str1 = string(rd.payload).substr(0,32)
    ;
135                             std::string str2 = string(rd.payload).substr
    (32,32);
136                             std::string str3 = string(rd.payload).substr
    (64,20);
137                             std::string str4 = string(rd.payload).substr
    (84,16);
138 
139                             std::string str5 = string(rd.payload).substr
    (100,16);
140 
141                             std::string str6 = string(rd.payload).substr
    (116,16);
142                             std::string str7 = string(rd.payload).substr
    (132,16);
143                             std::string str8 = string(rd.payload).substr
    (148,16);
144 
145                             std::string str9 = string(rd.payload).substr
    (164,32);
146                             std::string str10 = string(rd.payload).substr
    (196,32);
147                             std::string str11 = string(rd.payload).substr
    (228,32);
148 
149                             std::string str12 = string(rd.payload).substr
    (260,16);
150                             std::string str13 = string(rd.payload).substr
    (276,16);
151                             std::string str14 = string(rd.payload).substr
    (292,16);
152 
153                             // ACCORDING TO ICD
154                             double c1 = (double) ((binaryToNumeric<float>(str1
    , 32)) * (1 << 31));
155                             double c2 = (double) ((binaryToNumeric<float>(str2
    , 32)) * (1 << 31));
156 
157                             height = (int) ((binaryToNumeric<float>(str3, 20))
     * (1 << 3));
158 
159                             decToDMS(c1, &lon);
160                             decToDMS(c2, &lat);
161 
162                             speed = (unsigned int) ((binaryToNumeric<float>(
    str4, 16)) * (1 << 0));
163 
164                             heading = (float) ((binaryToNumeric<float>(str5,
    16)) * (1 << 15));  //in radians
165 
166                             pitch = (short) ((binaryToNumeric<float>(str6, 16)
    ) * (1 << 15));
167                             roll = (short) ((binaryToNumeric<float>(str7, 16))
     * (1 << 15));
168                             yaw = (short) ((binaryToNumeric<float>(str8, 16))
    * (1 << 15));
169 
170                             speed_NED.X = (double) ((binaryToNumeric<float>(
    str9, 32)) * (1 << 18));
171                             speed_NED.Y = (double) ((binaryToNumeric<float>(
    str10, 32)) * (1 << 18));
172                             speed_NED.Z = (double) ((binaryToNumeric<float>(
    str11, 32)) * (1 << 18));
173 
174                             acc_NED.X = (double) ((binaryToNumeric<float>(
    str12, 16)) * (1 << 5));
175                             acc_NED.Y = (double) ((binaryToNumeric<float>(
    str13, 16)) * (1 << 5));
176                             acc_NED.Z = (double) ((binaryToNumeric<float>(
    str14, 16)) * (1 << 5));
177 
```

```
178
179                             msg->pos.alt = height;
180                             msg->pos.lon = c1;
181                             msg->pos.lat = c2;
182                             msg->att.hd = heading;
183                             msg->att.pitch = pitch;
184                             msg->att.roll = roll;
185                             msg->att.yaw = yaw;
186                             msg->speed.X = speed_NED.X;
187                             msg->speed.Y = speed_NED.Y;
188                             msg->speed.Z = speed_NED.Z;
189
190                             msg->acc.X = acc_NED.X;
191                             msg->acc.Y = acc_NED.Y;
192                             msg->acc.Z = acc_NED.Z;
193
194                             msg->ac_speed = speed;
195
196                             msg->tmp = rd.received_time;
197
198                             //TODO!!! send backup packet over TS
199                         }
200                         else if(return_code==FACE::NOT_AVAILABLE){ //all data
    been transmitted
201                             break;
202                         }
203                         else{
204                             continue;
205                         }
206                     }
207
208                     start = 0;
209                     serialDPM.Close_Connection(handleDPM, return_code);
210                 }
211             }
212             else{
213                 start = time(0);
214             }
215 #endif
216
217                 break;
218             case FACE::INVALID_PARAM:
219                 break;
220             case FACE::INVALID_CONFIG:
221                 break;
222             case FACE::INVALID_MODE:
223                 break;
224             case FACE::TIMED_OUT:
225                 to_cnt++;
226                 if(to_cnt>2) {
227
228                     memset((void*) &msg_old, 0, sizeof(messaggio_EGI_t));
229
230                     perror("EGI_m=>Time out occurred!");
231                     msg_TS.timestamp = 0;
232                     FACE::TS::Send_Message(
233                             connId, FACE::INF_TIME_VALUE, txn, msg_TS,
234                             max_msg_size, status);
235
236                     printf("EGI Sending Message... %ld\n", msg_TS.timestamp);
237                     fflush(stdout);
238
239                     if (status != FACE::NO_ERROR) {
240                         perror("EGI_m=>Impossible to send the message!");
241                         return static_cast<int>(status);
242                     }
243                 }
244                 else{
245                     msg_TS.speed = msg_old.ac_speed;
246
247                     msg_TS.speed_NED.X = msg_old.speed.X;
248                     msg_TS.speed_NED.Y = msg_old.speed.Y;
249                     msg_TS.speed_NED.Z = msg_old.speed.Z;
250
251                     msg_TS.pos.lon = msg_old.pos.lon;
```

```
252                     msg_TS.pos.lat = msg_old.pos.lat;
253                     msg_TS.pos.alt = msg_old.pos.alt;
254
255                     msg_TS.att.roll = msg_old.att.roll;
256                     msg_TS.att.pitch = msg_old.att.pitch;
257                     msg_TS.att.yaw = msg_old.att.yaw;
258                     msg_TS.att.heading = msg_old.att.hd;
259
260                     msg_TS.acc_NED.X = msg_old.acc.X;
261                     msg_TS.acc_NED.Y = msg_old.acc.Y;
262                     msg_TS.acc_NED.Z = msg_old.acc.Z;
263
264                     msg_TS.timestamp = msg_old.tmp;
265
266                     FACE::TS::Send_Message(
267                             connId, FACE::INF_TIME_VALUE, txn, msg_TS,
268                             max_msg_size, status);
269
270                     /*printf("EGI Sending Message... %ld\n", msg_TS.timestamp)
    ;
271                     fflush(stdout);*/
272
273                     if (status != FACE::NO_ERROR) {
274                         perror("EGI_m=>Impossible to send the message!");
275                         return static_cast<int>(status);
276                     }
277                 }
278                 break;
279             case FACE::ADDR_IN_USE:
280                 break;
281             case FACE::PERMISSION_DENIED:
282                 break;
283             case FACE::MESSAGE_STALE:
284                 break;
285             case FACE::IN_PROGRESS:
286                 break;
287             case FACE::CONNECTION_CLOSED:
288                 break;
289             case FACE::DATA_BUFFER_TOO_SMALL:
290                 break;
291             case FACE::DATA_OVERFLOW:
292                 break;
293         }
294
295         auto end_mnt = std::chrono::high_resolution_clock::now();
296         auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(
    end_mnt - start_mnt).count();
297         if(duration>1e3){
298
299             mnt_msg.timestamp = time(nullptr);
300             mnt_msg.code = status;
301             //memcpy(wr.payload, (void*) &mnt_msg, sizeof(mnt_msg));
302
303             //printf("EGI to Monitor...\n");
304             serial_monitor.Write(serial_handle, FACE::IOSS::Serial::NO_WAIT, &
    wr, MONITOR_PACKET_SIZE, status);
305             //serial_monitor.Write(serial_handle, FACE::IOSS::Serial::
    WAIT_FOREVER, &wr, MONITOR_PACKET_SIZE, status);
306             if(status==FACE::NO_ERROR) {
307                 start_mnt = std::chrono::high_resolution_clock::now();
308             }
309             else{
310                 perror("ERR: EGI to Monitor");
311             }
312
313         }
314
315         //simulate sampling
316         usleep(delta_T_usec); //10ms
317
318     }
319
320     return FACE::NO_ERROR;
321 }
322
```

```
323 int EGI_manager::Close(){
324
325     FACE::RETURN_CODE_TYPE status;
326
327     myEGI.Close();
328
329     serial_monitor.Close_Connection(serial_handle, status);
330     if(status!=FACE::NO_ERROR){
331         perror("EGI=>Close connection error...");
332         return static_cast<int>(status);
333     }
334
335 #ifdef COMPRESS
336     free(uncmp_DPM_packet);
337 #endif
338
339     return FACE::NO_ERROR;
340
341 }
342
343 int EGI_manager::Destroy_TS_Connection(){
344     //ACE_OS::sleep(15);
345     FACE::RETURN_CODE_TYPE  status;
346
347     FACE::TS::Destroy_Connection(connId, status);
348
349     if (status != FACE::NO_ERROR) {
350         return static_cast<int>(status);
351     }
352
353     return FACE::NO_ERROR;
354 }
355
356 int EGI_manager::Flush(){
357     FACE::RETURN_CODE_TYPE status;
358     myEGI.Flush();
359     serial_monitor.Flush(serial_handle, status);
360     if(status!=FACE::NO_ERROR){
361         perror("EGI=>Close connection error...");
362         return static_cast<int>(status);
363     }
364
365     return FACE::NO_ERROR;
366 }
```

A.3.2.5 main.cpp

```
1  #include "EGI.h"
2  #include "EGI_manager.h"
3
4  //EGI_manager myEGI_man;
5
6  void signal_handler(int signal, siginfo_t *info, void *extra){
7      if(signal==SIGKILL){
8          printf("SIGKILL received!!!\n");
9  #ifdef COMPRESS
10         free(uncmp_DPM_packet);
11 #endif
12         EGI_manager::getMyEGI()->Destroy_TS_Connection();
13         EGI_manager::getMyEGI()->Flush();
14         EGI_manager::getMyEGI()->Close();
15     }
16     else{
17         perror("Another unknown signal detected!");
18     }
19 }
20
21 int main() {
22
23     struct sigaction sa;
24     sa.sa_sigaction = signal_handler;
```

```
    sa.sa_flags = SA_SIGINFO;
    sigemptyset(&sa.sa_mask);
    sigaction(SIGKILL, &sa, NULL);

    EGI_manager::getMyEGI()->Initialize();
    EGI_manager::getMyEGI()->Loop();
    EGI_manager::getMyEGI()->Destroy_TS_Connection();
    EGI_manager::getMyEGI()->Flush();
    EGI_manager::getMyEGI()->Close();

    return EXIT_SUCCESS;
}
```

A.3.3 GPS

A.3.3.1 GPS.h

```
//
// Created by Edoardo Cavallotti on 24/06/24.
//

#ifndef GPS_GPS_H
#define GPS_GPS_H

#include "FACE_LAYERS/IOSS/FACE_Serial/include/FACE_Serial.h"
#include "OpenDDS-3.28.1/FACE/common.hpp"
#include "../sim_utils.h"
#include "../DPM/DPM.h"
#include "../DPM/backup.h"
#include "AC_SIM/sim_constant.h"

class GPS {
public:
    void Initialize();
    void Open();
    void Read(messaggio_GPS_t *msg, FACE::RETURN_CODE_TYPE& return_status);
    void Close();
    void Flush();

private:
    FACE_Serial serial;
    FACE_Serial serialDPM;

    Coordinate_t lat;
    Coordinate_t lon;
    int height;
    float heading; // in radians
    short pitch;
    short roll;
    double speed;

    FACE::IOSS::Serial::CONNECTION_NAME_TYPE name;
    FACE::IOSS::CONNECTION_HANDLE_TYPE handle;

    FACE::IOSS::Serial::READ_PAYLOAD_TYPE rd;

    FACE::IOSS::Serial::CONNECTION_NAME_TYPE nameDPM;
    FACE::IOSS::CONNECTION_HANDLE_TYPE handleDPM;

#ifdef COMPRESS
    char *uncmp_packet;
#endif
};


#endif //GPS_GPS_H
```

A.3.3.2 GPS.cpp

```
//
// Created by Edoardo Cavallotti on 24/06/24.
//

#include "GPS.h"

void GPS::Initialize(){

    lat.degree = 0;
    lat.min = 0;
    lat.sec = 0;

    lon.degree = 0;
    lon.min = 0;
    lon.sec = 0;

    height = 0;
    heading = 0.0f;

    rd.payload = (FACE::IOSS::Serial::PAYLOAD_DATA_MSG_TYPE) malloc(sizeof(
    char)*GPS_PACKET_SIZE);

    FACE::RETURN_CODE_TYPE return_code;
    FACE::CONFIGURATION_RESOURCE config = "/Users/edoardocavallotti/Desktop/
    TESI/FACE_SOFTWARE_IMPLEMENTATION/FACE_LAYERS/PSSS/GPS/serialSIM";
    serial.Initialize(config, return_code);
    if(return_code!=FACE::NO_ERROR){
        perror("GPS=>Error!!");
    }

#ifdef COMPRESS
    uncmp_packet = (char*) malloc(sizeof(char)*GPS_PACKET_SIZE*8);
#endif

}

void GPS::Open() {

    FACE::RETURN_CODE_TYPE return_code;

    serial.Open_Connection(name, FACE::IOSS::Serial::WAIT_FOREVER, handle,
    return_code);
    if(return_code!=FACE::NO_ERROR){
        perror("GPS=>Error!!");
    }

}

void GPS::Read(messaggio_GPS_t *msg, FACE::RETURN_CODE_TYPE& return_status){

    //int fd = open("/Users/edoardocavallotti/Desktop/TESI/
    FACE_SOFTWARE_IMPLEMENTATION/FACE_LAYERS/PCS/GPS.txt", O_RDWR | O_TRUNC |
    O_CREAT);

    serial.Read(handle, FACE::IOSS::Serial::NO_WAIT, &rd, GPS_PACKET_SIZE,
    return_status);
    //serial.Read(handle, 1e8, &rd, GPS_PACKET_SIZE, return_status);


    switch(return_status){
        case FACE::NO_ERROR:{

#ifdef COMPRESS

            uncompress(uncmp_packet, rd.payload, GPS_PACKET_SIZE);

            std::string str1 = string(uncmp_packet).substr(0, 32);
            std::string str2 = string(uncmp_packet).substr(32, 32);
            std::string str3 = string(uncmp_packet).substr(64, 20);
            std::string str4 = string(uncmp_packet).substr(84, 16);
            std::string str5 = string(uncmp_packet).substr(100,16);
#else

```

```
68              std::string str1 = string(rd.payload).substr(0, 32);
69              std::string str2 = string(rd.payload).substr(32, 32);
70              std::string str3 = string(rd.payload).substr(64, 20);
71              std::string str4 = string(rd.payload).substr(84, 16);
72              std::string str5 = string(rd.payload).substr(100,16);
73  #endif
74
75              // ACCORDING TO ICD
76              double lon_d = (binaryToNumeric<double>(str1, 32)) * (1 << 31);
77              double lat_d = (binaryToNumeric<double>(str2, 32)) * (1 << 31);
78              height = ((binaryToNumeric<float>(str3, 20)) * (1 << 3));
79              speed = (unsigned int) ((binaryToNumeric<float>(str4, 16)) * (1 <<
     0));
80              heading = (float) ((binaryToNumeric<float>(str5, 16)) * (1 << 15))
    ;
81
82
83              decToDMS(lat_d, &lat);
84              decToDMS(lon_d, &lon);
85
86              msg->pos.alt = height;
87              msg->pos.lat = lat_d;
88              msg->pos.lon = lon_d;
89              msg->att.hd = heading;
90              msg->speed = speed;
91
92              break;
93          }
94          case FACE::NO_ACTION:
95              perror("GPS=>");
96              break;
97          case FACE::NOT_AVAILABLE:{
98              perror("GPS=>");
99              break;
100         }
101         case FACE::INVALID_PARAM:
102             perror("GPS=>");
103             break;
104         case FACE::INVALID_CONFIG:
105             perror("GPS=>");
106             break;
107         case FACE::INVALID_MODE:
108             perror("GPS=>");
109             break;
110         case FACE::TIMED_OUT:
111             perror("GPS=>Time out occurred!");
112             msg->tmp = 0; //flag to signal no packet has arrived
113             break;
114         case FACE::ADDR_IN_USE:
115             perror("GPS=>");
116             break;
117         case FACE::PERMISSION_DENIED:
118             perror("GPS=>");
119             break;
120         case FACE::MESSAGE_STALE:
121             perror("GPS=>");
122             break;
123         case FACE::IN_PROGRESS:
124             perror("GPS=>");
125             break;
126         case FACE::CONNECTION_CLOSED:
127             perror("GPS=>");
128             break;
129         case FACE::DATA_BUFFER_TOO_SMALL:
130             perror("GPS=>");
131             break;
132         case FACE::DATA_OVERFLOW:
133             perror("GPS=>");
134             break;
135     }
136
137     msg->tmp = rd.received_time;
138     //write(fd, (void*) &msg, sizeof(messaggio_GPS_t));
139     //fsync(fd);
140
```

```
141
142 }
143
144 void GPS::Close() {
145     FACE::RETURN_CODE_TYPE return_code;
146     serial.Close_Connection(handle, return_code);
147 #ifdef COMPRESS
148     free(uncmp_packet);
149 #endif
150 }
151
152 void GPS::Flush() {
153     FACE::RETURN_CODE_TYPE return_code;
154     serial.Flush(handle, return_code);
155 }
```

A.3.3.3 GPS_manager.h

```
1  //
2  // Created by Edoardo Cavallotti on 03/11/24.
3  //
4
5  #ifndef PSSS_GPS_MANAGER_H
6  #define PSSS_GPS_MANAGER_H
7
8  #include <iostream>
9  #include "unistd.h"
10 #include "fcntl.h"
11 #include <cmath>
12 #include "GPS.h"
13
14 #include "FACE_LAYERS/TSS/TS/FaceMessage_TS.hpp"
15 #include "ace/Log_Msg.h"
16 #include "ace/OS_NS_unistd.h"
17
18 #include <chrono>
19 #include <thread>
20 #include <signal.h>
21
22 #define MONITOR_PACKET_SIZE 16
23
24 #ifdef ACE_AS_STATIC_LIBS
25 # include "dds/DCPS/RTPS/RtpsDiscovery.h"
26 # include "dds/DCPS/transport/rtps_udp/RtpsUdp.h"
27 #endif
28
29 typedef struct monitor_packet_s{
30     long long timestamp;
31     int code;
32 }monitor_packet_t;
33
34 class GPS_manager{
35 public:
36
37     static GPS_manager* getMyGPS();
38
39     int Initialize();
40     int Loop();
41     int Close();
42     int Destroy_TS_Connection();
43     int Flush();
44 private:
45
46     static GPS_manager *myGPS_man;
47     GPS_manager();
48     ~GPS_manager();
49
50     GPS_manager(const GPS_manager&) = delete;
51     GPS_manager& operator=(const GPS_manager&) = delete;
52
53
```

```
    GPS myGPS;

    FACE::IOSS::Serial::CONNECTION_NAME_TYPE serial_name;
    FACE::IOSS::Serial::WRITE_PAYLOAD_TYPE wr;

    FACE_Serial serial_monitor;
    FACE::IOSS::CONNECTION_HANDLE_TYPE serial_handle;
    FACE::CONNECTION_ID_TYPE connId;

    FACE::CONNECTION_DIRECTION_TYPE dir;
    FACE::MESSAGE_SIZE_TYPE max_msg_size;
    FACE::TRANSACTION_ID_TYPE txn;

    monitor_packet_t mnt_msg;

    // Message to send
    Messenger::Position_GPS msg_TS;

    long start;
    short to_cnt;

    messaggio_GPS_t msg;
    messaggio_GPS_t msg_old;

};

#endif //PSSS_GPS_MANAGER_H
```

A.3.3.4 GPS_manager.cpp

```
//
// Created by Edoardo Cavallotti on 03/11/24.
//


#include "GPS_manager.h"

GPS_manager* GPS_manager::myGPS_man = nullptr;

GPS_manager::GPS_manager(){}

GPS_manager::~GPS_manager(){}

GPS_manager* GPS_manager::getMyGPS() {
    if (myGPS_man == nullptr) {
        myGPS_man = new GPS_manager();
    }
    return myGPS_man;
}

int GPS_manager::Initialize(){
    FACE::RETURN_CODE_TYPE status;

    FACE::CONFIGURATION_RESOURCE config = "/Users/edoardocavallotti/Desktop/
    TESI/FACE_SOFTWARE_IMPLEMENTATION/FACE_LAYERS/PSSS/GPS/GPS_ser_monitor";
    serial_monitor.Initialize(config, status);
    serial_monitor.Open_Connection(serial_name, FACE::IOSS::Serial::
    WAIT_FOREVER, serial_handle, status);
    wr.payload = (FACE::IOSS::Serial::PAYLOAD_DATA_MSG_TYPE) malloc(sizeof(
    monitor_packet_t));

    FACE::TS::Initialize("/Users/edoardocavallotti/Desktop/TESI/
    FACE_SOFTWARE_IMPLEMENTATION/FACE_LAYERS/TSS/TS/config_static_GPS.ini",
    status);

    if (status != FACE::NO_ERROR) {
        return static_cast<int>(status);
    }

    myGPS.Initialize();
    myGPS.Open();
```

```
37
38      // Create the pub connection
39      FACE::TS::Create_Connection(
40              "pub_GPS", FACE::PUB_SUB, connId, dir,
41              max_msg_size, FACE::INF_TIME_VALUE, status);
42
43      if (status != FACE::NO_ERROR) {
44          return static_cast<int>(status);
45      }
46
47  #ifdef COMPRESS
48      uncmp_DPM_packet = (char*) malloc(sizeof(char)*EGI_PACKET_SIZE*8);
49  #endif
50
51      // Send message
52
53      ACE_DEBUG((LM_INFO, "Publisher: about to Send_Message()\n"));
54
55      start = 0;
56      to_cnt = 0;
57
58  }
59
60  int GPS_manager::Loop(){
61
62      auto start_mnt = std::chrono::high_resolution_clock::now();
63      FACE::RETURN_CODE_TYPE status;
64
65      while(1){
66          memset((void*)&msg, 0, sizeof(messaggio_GPS_t));
67          memset((void*) &mnt_msg, 0, sizeof(monitor_packet_t));
68          myGPS.Read(&msg, status);
69          switch(status){
70              case FACE::NO_ERROR:
71                  to_cnt = 0;
72
73                  msg_TS.speed = msg.speed;
74
75                  msg_TS.pos.lon = msg.pos.lon;
76                  msg_TS.pos.lat = msg.pos.lat;
77                  msg_TS.pos.alt = msg.pos.alt;
78
79                  msg_TS.att.heading = msg.att.hd;
80
81                  msg_TS.timestamp = msg.tmp;
82
83                  FACE::TS::Send_Message(
84                          connId, FACE::INF_TIME_VALUE, txn, msg_TS,
85                          max_msg_size, status);
86
87                  if (status != FACE::NO_ERROR) {
88                      perror("GPS_m=>");
89                      return static_cast<int>(status);
90                  }
91
92                  msg_old = msg;
93
94                  break;
95              case FACE::NO_ACTION:
96                  break;
97              case FACE::NOT_AVAILABLE:
98  #ifdef SERVER_BACKUP
99                  if(start>0){
100                 if(time(0)-start>10){
101
102                     // prepare connection resources
103                     FACE::CONFIGURATION_RESOURCE config = "/Users/
    edoardocavallotti/Desktop/TESI/FACE_SOFTWARE_IMPLEMENTATION/FACE_LAYERS/
    PSSS/GPS/serialDPM";
104                     serialDPM.Initialize(config, return_code);
105                     if(return_code!=FACE::NO_ERROR){
106                         perror("GPS=>Error!!");
107                     }
108
109                     //open connection
```

```
110                     serialDPM.Open_Connection(nameDPM, FACE::IOSS::Serial::
    WAIT_FOREVER, handleDPM, return_code);
111                     if(return_code!=FACE::NO_ERROR){
112                         perror("GPS=>Error!!");
113                     }
114
115                     FACE::IOSS::Serial::WRITE_PAYLOAD_TYPE wr;
116                     wr.payload = (FACE::IOSS::Serial::PAYLOAD_DATA_MSG_TYPE)
    malloc(sizeof(char)*1);
117                     char ok = (char) 1;
118                     memcpy(wr.payload, (void*) &ok, sizeof(char));
119                     serialDPM.Write(handleDPM, FACE::IOSS::Serial::NO_WAIT, &
    wr, 1, return_code);  //to unlock the DPM service
120                     while(1){ // read data from DPM
121                         serialDPM.Read(handleDPM, FACE::IOSS::Serial::
    WAIT_FOREVER, &rd, GPS_PACKET_SIZE, return_code);
122                         if(return_code==FACE::NO_ERROR) {
123                             std::string str1 = string(rd.payload).substr(0,
    32);
124                             std::string str2 = string(rd.payload).substr(32,
    32);
125                             std::string str3 = string(rd.payload).substr(64,
    20);
126                             std::string str4 = string(rd.payload).substr(84,
    16);
127                             std::string str5 = string(rd.payload).substr
    (100,16);
128
129                             double lon_d = (binaryToNumeric<double>(str1, 32))
     * (1 << 31);
130                             double lat_d = (binaryToNumeric<double>(str2, 32))
     * (1 << 31);
131                             height = ((binaryToNumeric<float>(str3, 20)) * (1
    << 3));
132                             speed = (unsigned int) ((binaryToNumeric<float>(
    str4, 16)) * (1 << 0));
133                             heading = (float) ((binaryToNumeric<float>(str5,
    16)) * (1 << 15));
134
135
136                             decToDMS(lat_d, &lat);
137                             decToDMS(lon_d, &lon);
138
139                             msg->pos.alt = height;
140                             msg->pos.lat = lat_d;
141                             msg->pos.lon = lon_d;
142                             msg->att.hd = heading;
143                             msg->speed = speed;
144
145                             //TODO!! send_data_over_TS
146
147                         }
148                         else if(return_code==FACE::NOT_AVAILABLE){ //all data
    been transmitted
149                             break;
150                         }
151                         else{
152                             continue;
153                         }
154                     }
155
156                     start = 0;
157                     serialDPM.Close_Connection(handleDPM, return_code);
158                 }
159             }
160             else{
161                 start = time(0);
162             }
163 #endif
164                 break;
165             case FACE::INVALID_PARAM:
166                 break;
167             case FACE::INVALID_CONFIG:
168                 break;
169             case FACE::INVALID_MODE:
```

```
170                 break;
171             case FACE::TIMED_OUT:
172                 to_cnt++;
173                 if(to_cnt>2) {
174 
175                     memset((void*) &msg_old, 0, sizeof(messaggio_GPS_t));
176 
177                     perror("GPS_m=>Time out occurred!");
178                     msg_TS.timestamp = 0;
179                     FACE::TS::Send_Message(
180                             connId, FACE::INF_TIME_VALUE, txn, msg_TS,
181                             max_msg_size, status);
182 
183                     /*printf("EGI Sending Message... %ld\n", msg_TS.timestamp)
    ;
184                     fflush(stdout);*/
185 
186                     if (status != FACE::NO_ERROR) {
187                         perror("GPS_m=>Impossible to send the message!");
188                         return static_cast<int>(status);
189                     }
190                 }
191                 else{
192                     msg_TS.speed = msg_old.speed;
193                     msg_TS.pos.lon = msg_old.pos.lon;
194                     msg_TS.pos.lat = msg_old.pos.lat;
195                     msg_TS.pos.alt = msg_old.pos.alt;
196 
197                     msg_TS.att.heading = msg_old.att.hd;
198 
199                     msg_TS.timestamp = msg_old.tmp;
200 
201                     FACE::TS::Send_Message(
202                             connId, FACE::INF_TIME_VALUE, txn, msg_TS,
203                             max_msg_size, status);
204 
205                     /*printf("GPS Sending Message... %ld\n", msg_TS.timestamp)
    ;
206                     fflush(stdout);*/
207 
208                     if (status != FACE::NO_ERROR) {
209                         perror("GPS_m=>");
210                         return static_cast<int>(status);
211                     }
212                 }
213 
214                 break;
215             case FACE::ADDR_IN_USE:
216                 break;
217             case FACE::PERMISSION_DENIED:
218                 break;
219             case FACE::MESSAGE_STALE:
220                 break;
221             case FACE::IN_PROGRESS:
222                 break;
223             case FACE::CONNECTION_CLOSED:
224                 break;
225             case FACE::DATA_BUFFER_TOO_SMALL:
226                 break;
227             case FACE::DATA_OVERFLOW:
228                 break;
229         }
230         auto end_mnt = std::chrono::high_resolution_clock::now();
231         auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(
    end_mnt - start_mnt).count();
232         if(duration>1e3){
233 
234             mnt_msg.timestamp = time(nullptr);
235             mnt_msg.code = status;
236             //memcpy(wr.payload, (void*) &mnt_msg, sizeof(mnt_msg));
237 
238             //printf("GPS to Monitor...\n");
239              serial_monitor.Write(serial_handle, FACE::IOSS::Serial::
    WAIT_FOREVER, &wr, MONITOR_PACKET_SIZE, status);
240             if(status==FACE::NO_ERROR) {
```

```
241                 start_mnt = std::chrono::high_resolution_clock::now();
242             }
243             else{
244                 perror("ERR: GPS to Monitor");
245             }
246 
247         }
248 
249         //simulate sampling
250         usleep(delta_T_usec); //10ms
251 
252     }
253 
254     return FACE::NO_ERROR;
255 }
256 
257 int GPS_manager::Close(){
258 
259     FACE::RETURN_CODE_TYPE status;
260 
261     myGPS.Close();
262 
263     serial_monitor.Close_Connection(serial_handle, status);
264     if(status!=FACE::NO_ERROR){
265         perror("GPS=>Close connection error...");
266         return static_cast<int>(status);
267     }
268 
269 #ifdef COMPRESS
270     free(uncmp_DPM_packet);
271 #endif
272 
273     return FACE::NO_ERROR;
274 
275 }
276 
277 int GPS_manager::Destroy_TS_Connection(){
278     //ACE_OS::sleep(15);
279     FACE::RETURN_CODE_TYPE  status;
280 
281     FACE::TS::Destroy_Connection(connId, status);
282 
283     if (status != FACE::NO_ERROR) {
284         return static_cast<int>(status);
285     }
286 
287     return FACE::NO_ERROR;
288 }
289 
290 int GPS_manager::Flush(){
291     FACE::RETURN_CODE_TYPE status;
292     myGPS.Flush();
293     serial_monitor.Flush(serial_handle, status);
294     if(status!=FACE::NO_ERROR){
295         perror("GPS=>Close connection error...");
296         return static_cast<int>(status);
297     }
298 
299     return FACE::NO_ERROR;
300 }
```

A.3.3.5 main.cpp

```
1 #include "GPS.h"
2 #include "GPS_manager.h"
3 
4 //GPS_manager myGPS_man;
5 
6 void signal_handler(int signal, siginfo_t *info, void *extra){
7     if(signal==SIGKILL){
8         printf("SIGKILL received!!!\n");
```

```
9
10 #ifdef COMPRESS
11         free(uncmp_DPM_packet);
12 #endif
13         GPS_manager::getMyGPS()->Destroy_TS_Connection();
14         GPS_manager::getMyGPS()->Flush();
15         GPS_manager::getMyGPS()->Close();
16     }
17     else{
18         perror("Another unknown signal detected!");
19     }
20 }
21
22 int main() {
23
24     struct sigaction sa;
25     sa.sa_sigaction = signal_handler;
26     sa.sa_flags = SA_SIGINFO;
27     sigemptyset(&sa.sa_mask);
28     sigaction(SIGKILL, &sa, NULL);
29
30     GPS_manager::getMyGPS()->Initialize();
31     GPS_manager::getMyGPS()->Loop();
32     GPS_manager::getMyGPS()->Destroy_TS_Connection();
33     GPS_manager::getMyGPS()->Flush();
34     GPS_manager::getMyGPS()->Close();
35
36     return EXIT_SUCCESS;
37 }
```

A.3.4 Health Manager

A.3.4.1 main.cpp

```
1  #include <iostream>
2
3  #include "FACE_LAYERS/TSS/TS/FaceMessage_TS.hpp"
4  #include "FACE_LAYERS/IOSS/FACE_IO.h"
5  #include "ace/Log_Msg.h"
6  #include "ace/OS_NS_unistd.h"
7  #include "OpenDDS-3.28.1/FACE/common.hpp"
8  #include "FACE_LAYERS/IOSS/FACE_Serial/include/FACE_Serial.h"
9  #include "FACE_LAYERS/PCS/DF_utils.h"
10
11 #ifdef ACE_AS_STATIC_LIBS
12 # include "dds/DCPS/RTPS/RtpsDiscovery.h"
13 # include "dds/DCPS/transport/rtps_udp/RtpsUdp.h"
14 #endif
15
16 #define MONITOR_PACKET_SIZE 16
17
18 typedef struct monitor_packet_s{
19     long long timestamp;
20     int code;
21 }monitor_packet_t;
22
23 pid_t processes[10];
24 /*
25  * 0 => HM
26  * 1 => DF
27  * 2 => EGI
28  * 3 => GPS
29  * 4 => AC_SIM
30  * 5 => AC_POS
31  * 6 => socat1
32  * 7 => socat2
33  * 8 => socat3
34  * 9 => socat4
35  */
36
```

```
int main() {

    // remove old contents
    int pids_file_desc = open("FACE_LAYERS/pids.txt", O_TRUNC | O_WRONLY);
    close(pids_file_desc);

    // serial
    if(fork()==0){
        printf("Launching serial.sh\n");
        execl("FACE_LAYERS/serial.sh", "SERIAL", NULL);
    }

    sleep(2);
    //printf("P\n");

    //pid_t processes[10];

    // saving serial's pids (socat)
    FILE *pids_file = fopen("FACE_LAYERS/pids.txt", "r");
    for(short i=0; i<4; i++){
        fscanf(pids_file, "%d\n", &processes[i+6]);
    }
    fclose(pids_file);

    // saving HM's pid
    processes[0] = getpid();

    // saving other pids
    if((processes[4]=fork())==0){
        printf("Launching AC_SIM...\n");
        fflush(stdout);
        execl("AC_SIM/build/AC_SIM",
               "AC_SIM",
               NULL);
    }
    else{

    }

    if((processes[5]=fork())==0){
        printf("Launching AC_POSITION_APP...\n");
        fflush(stdout);
        execl("FACE_LAYERS/PSSS/AC_POS_APP_GRAPHIC/build/AC_POS_APP_GRAPHIC",
               "AC_POSITION_APP",
               NULL);
    }
    else{

    }


    // LAUNCH DF
    if((processes[1]=fork())==0){
        printf("Launching DF...\n");
        fflush(stdout);
        execl("FACE_LAYERS/PCS/DF/build/DF",
               "DF",
               NULL);
    }
    else{

    }

    // LAUNCH EGI
    if((processes[2]=fork())==0){
        printf("Launching EGI...\n");
        fflush(stdout);
        execl("FACE_LAYERS/PSSS/EGI/build/EGI",
               "EGI",
               NULL);
    }
    else{

    }

```

```
112     // LAUNCH GPS
113     if((processes[3]=fork())==0){
114         printf("Launching GPS...\n");
115         fflush(stdout);
116         execl("FACE_LAYERS/PSSS/GPS/build/GPS",
117                "GPS",
118                NULL);
119     }
120     else{
121 
122     }
123 
124     // flush pids file
125 
126 
127     // saving ALL the pids
128     pids_file = fopen("FACE_LAYERS/pids.txt", "w");
129     for(short i=0; i<10; i++){
130         fprintf(pids_file, "%d\n", processes[i]);
131     }
132     fclose(pids_file);
133 
134 
135     // from now on this code is correct
136     FACE::IOSS::Serial::CONNECTION_NAME_TYPE name_egi;
137     FACE::IOSS::Serial::CONNECTION_NAME_TYPE name_gps;
138     FACE::IOSS::Serial::CONNECTION_NAME_TYPE name_acpa;
139     FACE::IOSS::CONNECTION_HANDLE_TYPE handle_egi;
140     FACE::IOSS::CONNECTION_HANDLE_TYPE handle_gps;
141     FACE::IOSS::CONNECTION_HANDLE_TYPE handle_acpa;
142 
143     FACE::RETURN_CODE_TYPE status;
144 
145     monitor_packet_t egi_mnt;
146     monitor_packet_t gps_mnt;
147     Messenger::Monitor msg_TS_monitor;
148     FACE::IOSS::Serial::READ_PAYLOAD_TYPE rd;
149     rd.payload = (FACE::IOSS::Serial::PAYLOAD_DATA_MSG_TYPE) malloc(
    MONITOR_PACKET_SIZE);
150 
151     FACE::IOSS::Serial::READ_PAYLOAD_TYPE rd_acpa;
152     FACE::IOSS::Serial::WRITE_PAYLOAD_TYPE wr_acpa;
153 
154     FACE::RETURN_CODE_TYPE status_monitor;
155 
156     printf("FACE::TS Initialization...\n");
157     FACE::TS::Initialize("FACE_LAYERS/TSS/TS/config_static_HM.ini", status);
158 
159     if (status != FACE::NO_ERROR) {
160         return static_cast<int>(status);
161     }
162 
163     printf("FACE::TS Connection...\n");
164 
165     FACE::CONNECTION_ID_TYPE connId_mnt;
166     FACE::CONNECTION_DIRECTION_TYPE dir_mnt;
167     FACE::MESSAGE_SIZE_TYPE max_msg_size_mnt;
168     // Connection to monitor the DF behaviour
169     FACE::TS::Create_Connection(
170             "sub_Monitor", FACE::PUB_SUB, connId_mnt, dir_mnt,
171             max_msg_size_mnt, FACE::INF_TIME_VALUE, status);
172 
173     if (status != FACE::NO_ERROR) {
174         perror("sub_monitor...");
175         return static_cast<int>(status);
176     }
177 
178     // Receive message
179     FACE::TRANSACTION_ID_TYPE txn_mnt;
180     FACE_Serial ser_egi;
181     FACE_Serial ser_gps;
182     FACE_Serial ser_acpa;
183 
184     FACE::CONFIGURATION_RESOURCE config_egi = "FACE_LAYERS/PSSS/HEALTH_MANAGER
    /EGI_ser_monitor";
```

```
185     ser_egi.Initialize(config_egi, status);
186     if(status!=FACE::NO_ERROR){
187         perror("HM=>Error!!");
188     }
189     ser_egi.Open_Connection(name_egi, FACE::IOSS::Serial::WAIT_FOREVER,
        handle_egi, status);
190     if(status!=FACE::NO_ERROR){
191         perror("HM=>Error!!");
192     }
193
194     FACE::CONFIGURATION_RESOURCE config_gps = "FACE_LAYERS/PSSS/HEALTH_MANAGER
        /GPS_ser_monitor";
195     ser_gps.Initialize(config_gps, status);
196     if(status!=FACE::NO_ERROR){
197         perror("HM=>Error!!");
198     }
199     ser_gps.Open_Connection(name_gps, FACE::IOSS::Serial::WAIT_FOREVER,
        handle_gps, status);
200     if(status!=FACE::NO_ERROR){
201         perror("HM=>Error!!");
202     }
203
204     FACE::CONFIGURATION_RESOURCE config_acpa = "FACE_LAYERS/PSSS/
        HEALTH_MANAGER/ACPA_ser_monitor";
205     ser_acpa.Initialize(config_acpa, status);
206     if(status!=FACE::NO_ERROR){
207         perror("HM=>Error!!");
208     }
209     ser_acpa.Open_Connection(name_acpa, FACE::IOSS::Serial::WAIT_FOREVER,
        handle_acpa, status);
210     if(status!=FACE::NO_ERROR){
211         perror("HM=>Error!!");
212     }
213
214     short cnt = 0;
215     short lost[3] = {0,0,0};
216
217     bool isSomeKilled = false;
218
219     FACE::HM_APP_t hm_status;
220     memset((void*) &hm_status, 0, sizeof(FACE::HM_APP_t));
221
222     FACE::HM_KILL_t kill_msg;
223     memset((void*) &kill_msg, 0, sizeof(FACE::HM_KILL_t));
224
225     rd_acpa.payload = (FACE::IOSS::Serial::PAYLOAD_DATA_MSG_TYPE) malloc(
        sizeof(FACE::HM_KILL_t));
226     wr_acpa.payload = (FACE::IOSS::Serial::PAYLOAD_DATA_MSG_TYPE) malloc(
        sizeof(FACE::HM_APP_t));
227
228     auto start = std::chrono::high_resolution_clock::now();
229
230     while(1){
231
232         isSomeKilled = false;
233
234         cnt = 0;
235
236         // Receiving from DF
237         /*FACE::TS::Receive_Message(
238                 connId_mnt, FACE::INF_TIME_VALUE, txn_mnt, msg_TS_monitor,
239                 max_msg_size_mnt, status_monitor);*/
240         FACE::TS::Receive_Message(
241                 connId_mnt, 1e9, txn_mnt, msg_TS_monitor,
242                 max_msg_size_mnt, status_monitor);
243         if (status != FACE::NO_ERROR) {
244             perror("HM... (DF)");
245             printf("%d\n", status);
246             //return static_cast<int>(status);
247             lost[0]++;
248         }
249         else{
250             cnt++;
251             lost[0] = 0;
252         }
```

```
253
254         // Receiving from EGI
255         ser_egi.Read(handle_egi, 1e9, &rd, MONITOR_PACKET_SIZE, status);
256         if (status != FACE::NO_ERROR) {
257             perror("HM... (EGI)");
258             printf("%d\n", status);
259             //return static_cast<int>(status);
260             lost[1]++;
261         }
262         else{
263             memcpy((void*) &egi_mnt, (void*) (rd.payload), sizeof(
    monitor_packet_t));
264             cnt++;
265             lost[1] = 0;
266         }
267
268         // Receiving from GPS
269
270         ser_gps.Read(handle_gps, 1e9, &rd, MONITOR_PACKET_SIZE, status);
271         if (status != FACE::NO_ERROR) {
272             perror("HM... (GPS)");
273             printf("%d\n", status);
274             //return static_cast<int>(status);
275             lost[2]++;
276         }
277         else{
278             memcpy((void*) &gps_mnt, (void*) (rd.payload), sizeof(
    monitor_packet_t));
279             cnt++;
280             lost[2] = 0;
281         }
282
283
284         // HM CHECKING
285         if(cnt==3){
286             printf("HM: all three packets received!!\n");
287         }
288         else{
289             perror("Message lost...");
290             if(lost[0]>10){ //DF
291                 perror("DF stopped working...");
292                 hm_status.df = 1;
293
294
295
296             }
297             else if(lost[1]>10){  //EGI
298                 perror("EGI stopped working...");
299                 hm_status.egi = 1;
300             }
301             else if(lost[2]>10){  //GPS
302                 perror("GPS stopped working...");
303                 hm_status.gps = 1;
304             }
305         }
306
307
308         auto end = std::chrono::high_resolution_clock::now();
309         auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(
    end - start).count();
310         if(duration>5e3){ // 5 seconds
311
312
313             memcpy((void*) wr_acpa.payload, (void*) &hm_status, sizeof(FACE::
    HM_APP_t));
314             ser_acpa.Write(handle_acpa, FACE::NO_WAIT, &wr_acpa, sizeof(FACE::
    HM_APP_t), status);
315
316             usleep(1000);
317
318             ser_acpa.Read(handle_acpa, FACE::NO_WAIT, &rd_acpa, sizeof(FACE::
    HM_KILL_t), status);
319             memcpy((void*) &kill_msg, (void*) rd_acpa.payload, sizeof(FACE::
    HM_KILL_t));
320
```

```
321
322             if(status==FACE::NO_ERROR){
323                 switch(kill_msg.option){
324                     case 1:{ //DF
325                         printf("Killing DF process...\n");
326                         FACE::TS::Destroy_Connection(connId_mnt, status);
327                         if(status!=FACE::NO_ERROR){
328                             perror("DF=>Close connection error...");
329                         }
330                         kill(processes[1], SIGKILL);
331                         printf("Killing DF\n");
332                         pid_t pid;
333                         printf("Launching DF\n");
334                         fflush(stdout);
335                         if((pid=fork())==0){//child
336                             execl("FACE_LAYERS/PCS/DF/build/DF", "DF", NULL);
337                             perror("OPS");
338                         }
339                         else{ //father continues
340                             processes[1] = pid;
341                             isSomeKilled = true;
342                         }
343                         lost[0]=0;
344
345                         FACE::TS::Create_Connection(
346                                 "sub_Monitor", FACE::PUB_SUB, connId_mnt,
    dir_mnt,
347                                 max_msg_size_mnt, FACE::INF_TIME_VALUE, status
    );
348
349                         if (status != FACE::NO_ERROR) {
350                             perror("EGI...");
351                             return static_cast<int>(status);
352                         }
353
354                         break;
355                     }
356                     case 2:{ //EGI
357                         printf("Killing EGI process...\n");
358                         ser_egi.Flush(handle_egi, status);
359                         ser_egi.Close_Connection(handle_egi, status);
360
361                         kill(processes[2], SIGKILL);
362
363                         pid_t pid;
364                         if((pid=fork())==0){//child
365                             execl("FACE_LAYERS/PSSS/EGI/build/EGI", "EGI",
    NULL);
366                             perror("");
367                         }
368                         else{ //father continues
369                             processes[2] = pid;
370                             isSomeKilled = true;
371                         }
372
373                         ser_egi.Initialize(config_egi, status);
374                         if(status!=FACE::NO_ERROR){
375                             perror("HM=>Error!!");
376                         }
377                         ser_egi.Open_Connection(name_egi, FACE::IOSS::Serial::
    WAIT_FOREVER, handle_egi, status);
378                         if(status!=FACE::NO_ERROR){
379                             perror("HM=>Error!!");
380                         }
381
382                         lost[1]=0;
383
384                         break;
385                     }
386                     case 3:{ //GPS
387                         printf("Killing GPS process...\n");
388                         ser_gps.Flush(handle_gps, status);
389                         ser_gps.Close_Connection(handle_gps, status);
390
391                         kill(processes[3], SIGKILL);
```

```
392
393                         pid_t pid;
394                         if((pid=fork())==0){//child
395                             execl("FACE_LAYERS/PSSS/GPS/build/GPS", "GPS",
    NULL);
396                             perror("");
397                         }
398                         else{ //father continues
399                             processes[3] = pid;
400                             isSomeKilled = true;
401                         }
402
403                         ser_gps.Initialize(config_gps, status);
404                         if(status!=FACE::NO_ERROR){
405                             perror("HM=>Error!!");
406                         }
407                         ser_gps.Open_Connection(name_gps, FACE::IOSS::Serial::
    WAIT_FOREVER, handle_gps, status);
408                         if(status!=FACE::NO_ERROR){
409                             perror("HM=>Error!!");
410                         }
411
412                         lost[2]=0;
413
414                         break;
415                     }
416                 }
417             }
418             else if(status==FACE::TIMED_OUT){
419                 printf("All ok!\n");
420             }
421
422             FACE::HM_APP_t hm_status;
423             memset((void*) &hm_status, 0, sizeof(FACE::HM_APP_t));
424
425             FACE::HM_KILL_t kill_msg;
426             memset((void*) &kill_msg, 0, sizeof(FACE::HM_KILL_t));
427
428             start = std::chrono::high_resolution_clock::now();
429         }
430
431
432         if(isSomeKilled){
433             pids_file = fopen("FACE_LAYERS/pids.txt", "w");
434             for(short i=0; i<10; i++){
435                 fprintf(pids_file, "%d\n", processes[i]);
436             }
437             fclose(pids_file);
438         }
439
440     }
441
442     ser_egi.Flush(handle_egi, status);
443     ser_egi.Close_Connection(handle_egi, status);
444     ser_gps.Flush(handle_gps, status);
445     ser_gps.Close_Connection(handle_gps, status);
446     ser_acpa.Flush(handle_acpa, status);
447     ser_acpa.Close_Connection(handle_acpa, status);
448
449     return 0;
450 }
```

A.3.5 PSGS Display

A.3.5.1 left_screen.h

```
1 #ifndef CUSTOMWIDGET_LEFT_H
2 #define CUSTOMWIDGET_LEFT_H
3
4 #include <QWidget>
```

```
#include <QTimer>
#include <QPainter>
#include <QLabel>
#include <QLineEdit>
#include <QSplitter>
#include <QVBoxLayout>
#include <QPushButton>
#include <QMenu>
#include <QMenuBar>
#include "map_panel.h"
#include "unistd.h"
#include "fcntl.h"
#include "FACE_LAYERS/IOSS/FACE_Serial/include/FACE_Serial.h"
#include "FACE_LAYERS/PSSS/sim_utils.h"
#include "FACE_LAYERS/IOSS/Common.h"
#include "OpenDDS-3.28.1/FACE/common.hpp"
#include "FACE_LAYERS/PCS/DF_utils.h"

#include "flag.h"

class PositionWidget : public QWidget
{
    Q_OBJECT

public:
    explicit PositionWidget(QWidget *parent = nullptr);
    void load_airplane();
    void load_rose();
    ~PositionWidget();

    double get_lat() const;
    double get_lon() const;
    int get_alt() const;
    float get_heading() const;
    short get_yaw();

    void set_lat(double lat);
    void set_lon(double lon);
    void set_alt(int alt);
    void set_heading(float heading);
    void set_yaw(short yaw);
    void set_lat(double lat, int opt);
    void set_lon(double lon, int opt);

    double get_ac_speed() const;
    void set_ac_speed(double ac_speed);
    speed_t get_speed() const;
    void set_speed(const speed_t speed);
    acc_t get_acc() const;
    void set_acc(const acc_t acc);

    void setEgiRead(bool);
    bool getEgiRead();

    void setGpsRead(bool);
    bool getGpsRead();

    void setEgiOn(bool);
    bool getEgiOn();

    void setGpsOn(bool);
    bool getGpsOn();

    void set_EGI_APP(bool);
    void set_GPS_APP(bool);
    void set_DF_APP(bool);

    bool get_EGI_APP();
    bool get_GPS_APP();
    bool get_DF_APP();

    void read_serial(char *buf, FACE::RETURN_CODE_TYPE &status);
    void write_serial(char *buf, FACE::RETURN_CODE_TYPE &status);


```

```
protected:
    void paintEvent(QPaintEvent *event) override;

public slots:
    void updatePosition();

private slots:
    void egi_switch_clicked();
    void gps_switch_clicked();

    void switch_DF_HM();
    void switch_GPS_HM();
    void switch_EGI_HM();

    void send_DF_kill();
    void send_GPS_kill();
    void send_EGI_kill();

private:
    void setupUI();

    QString double2dms_string(double);

    QTimer *timer;
    int offset;
    int plane[35][33];
    int rose[66][60];
    int Cx = 17;
    int Cy = 16;
    QLabel *lat_l;
    QLabel *lon_l;
    QLabel *alt_l;

    QLineEdit *lat_t;
    QLineEdit *lon_t;
    QLineEdit *alt_t;

    QLabel *head_l;
    QLineEdit *head_t;

    QLabel *ac_speed_l;
    QLineEdit *ac_speed_t;

    QLabel *speed_x_l;
    QLabel *speed_y_l;
    QLabel *speed_z_l;

    QLineEdit *speed_x;
    QLineEdit *speed_y;
    QLineEdit *speed_z;

    QLabel *acc_x_l;
    QLabel *acc_y_l;
    QLabel *acc_z_l;

    QLineEdit *acc_x;
    QLineEdit *acc_y;
    QLineEdit *acc_z;

    QVBoxLayout *layout;
    QWidget *topPanel;
    MapPanel *map;

    QLabel *status_egi;
    QLabel *status_gps;

    QPushButton *egi_switch;
    QPushButton *gps_switch;

    QPushButton *btn_DF;
    QPushButton *btn_EGI;
    QPushButton *btn_GPS;

    double lat;
    double lon;
```

```
155     int alt;
156     float heading;
157     short yaw;
158 
159     double lat_egi;
160     double lon_egi;
161     double lat_gps;
162     double lon_gps;
163 
164     double ac_speed;
165     speed_t speed;
166     acc_t acc;
167 
168     bool isEgiRead;
169     bool isGpsRead;
170 
171     bool isEgiOn;
172     bool isGpsOn;
173 
174     bool EGI_APP;
175     bool GPS_APP;
176     bool DF_APP;
177 
178     bool EGI_APP_pressed;
179     bool GPS_APP_pressed;
180     bool DF_APP_pressed;
181 
182     FACE::RETURN_CODE_TYPE status;
183 
184 #ifdef USE_TSS
185     // to HM via Serial
186     FACE_Serial ser_hm;
187     FACE::IOSS::Serial::CONNECTION_NAME_TYPE name_hm;
188     FACE::IOSS::CONNECTION_HANDLE_TYPE handle_hm;
189     FACE::IOSS::Serial::READ_PAYLOAD_TYPE rd;
190     FACE::IOSS::Serial::WRITE_PAYLOAD_TYPE wr;
191     FACE::HM_KILL_t kill_msg;
192 #endif
193 
194 };
195 
196 
197 
198 #endif // CUSTOMWIDGET_LEFT_H
```

A.3.5.2 left_screen.cpp

```
1  #include "left_screen.h"
2  #include "errno.h"
3  
4  //#define BUFFER_SIZE 13
5  
6  PositionWidget::PositionWidget(QWidget *parent)
7      : QWidget(parent)
8  {
9      offset = 0;
10     load_airplane();
11     load_rose();
12 
13     setupUI();
14     map->setPlane(this->plane);
15     map->setRose(this->rose);
16 
17 #ifdef USE_TSS
18     //HM serial connection
19     FACE::CONFIGURATION_RESOURCE config_hm = "FACE_LAYERS/PSSS/
       AC_POS_APP_GRAPHIC/HM_ser_monitor";
20     ser_hm.Initialize(config_hm, status);
21     if(status!=FACE::NO_ERROR){
22         perror("ACPA=>Error!!");
23     }
```

```
24      ser_hm.Open_Connection(name_hm, FACE::IOSS::Serial::WAIT_FOREVER,
        handle_hm, status);
25      if(status!=FACE::NO_ERROR){
26          perror("ACPA=>Error!!");
27      }
28
29      rd.payload = (FACE::IOSS::Serial::PAYLOAD_DATA_MSG_TYPE) malloc(sizeof(
        FACE::HM_APP_t));
30      wr.payload = (FACE::IOSS::Serial::PAYLOAD_DATA_MSG_TYPE) malloc(sizeof(
        FACE::HM_KILL_t));
31
32  #endif
33
34  }
35
36  void PositionWidget::setupUI(){
37
38      isEgiRead = true;
39      isGpsRead = true;
40      isEgiOn = true;
41      isGpsOn = true;
42
43      DF_APP_pressed = false;
44      EGI_APP_pressed = false;
45      GPS_APP_pressed = false;
46
47      DF_APP = true;
48      EGI_APP = true;
49      GPS_APP = true;
50
51  #ifdef USE_TSS
52      kill_msg.option = 0;
53  #endif
54
55      layout = new QVBoxLayout(this);
56
57      lat_l = new QLabel("LAT: ", this);
58      lon_l = new QLabel("LON: ", this);
59      alt_l = new QLabel("ALT: ", this);
60
61      lat_t = new QLineEdit("");
62      lon_t = new QLineEdit("");
63      alt_t = new QLineEdit("");
64
65      lat_t->setReadOnly(true);
66      lon_t->setReadOnly(true);
67      alt_t->setReadOnly(true);
68
69      lat_t->setFixedWidth(100);
70      lon_t->setFixedWidth(100);
71      alt_t->setFixedWidth(100);
72
73      head_l = new QLabel("HEADING: ", this);
74      head_t = new QLineEdit("");
75      head_l->setFixedWidth(80);
76
77      head_t->setReadOnly(true);
78      head_t->setFixedWidth(80);
79
80      ac_speed_l = new QLabel("A/C SPEED", this);
81      ac_speed_t = new QLineEdit("", this);
82
83      ac_speed_t->setReadOnly(true);
84      ac_speed_t->setFixedWidth(80);
85      ac_speed_l->setFixedWidth(80);
86
87      speed_x_l = new QLabel("SPEED X", this);
88      speed_y_l = new QLabel("SPEED Y", this);
89      speed_z_l = new QLabel("SPEED Z", this);
90
91      speed_x = new QLineEdit("");
92      speed_y = new QLineEdit("");
93      speed_z = new QLineEdit("");
94
95      speed_x->setReadOnly(true);
```

```
96      speed_y->setReadOnly(true);
97      speed_z->setReadOnly(true);
98
99      speed_x->setFixedWidth(70);
100     speed_y->setFixedWidth(70);
101     speed_z->setFixedWidth(70);
102
103     acc_x_l = new QLabel("ACC X", this);
104     acc_y_l = new QLabel("ACC Y", this);
105     acc_z_l = new QLabel("ACC Z", this);
106
107     acc_x = new QLineEdit("");
108     acc_y = new QLineEdit("");
109     acc_z = new QLineEdit("");
110
111     acc_x->setReadOnly(true);
112     acc_y->setReadOnly(true);
113     acc_z->setReadOnly(true);
114
115     acc_x->setFixedWidth(70);
116     acc_y->setFixedWidth(70);
117     acc_z->setFixedWidth(70);
118
119     QFont font;
120     QLabel *data_frame_label = new QLabel("AIRCRAFT DATA", this);
121     font.setBold(true); data_frame_label->setFont(font);
122
123     QLabel *map_frame_label = new QLabel("2D MAP A/C POSITION", this);
124     font.setBold(true); map_frame_label->setFont(font);
125     map_frame_label->setFixedHeight(20);
126
127     QFrame *f1 = new QFrame(this);
128     QFrame *f2 = new QFrame(this);
129
130     status_egi = new QLabel("EGI SENS", this);
131     status_gps = new QLabel("GPS SENS", this);
132
133     egi_switch = new QPushButton("", this);
134     gps_switch = new QPushButton("", this);
135
136     if(isEgiOn){
137         egi_switch->setStyleSheet(
138             "background-color: lightgrey; "
139             "color: black; "
140             //"font-weight: bold; "
141             "border-radius: 5px; "
142             "padding: 7px;"
143             );
144         egi_switch->setText("EGI");
145     }
146     else{
147         egi_switch->setStyleSheet(
148             "background-color: darkgrey; "
149             "color: black; "
150             //"font-weight: bold; "
151             "border-radius: 5px; "
152             "padding: 7px;"
153             );
154         egi_switch->setText("EGI");
155     }
156
157     if(isGpsOn){
158         gps_switch->setStyleSheet(
159             "background-color: lightgrey; "
160             "color: black; "
161             //"font-weight: bold; "
162             "border-radius: 5px; "
163             "padding: 7px;"
164             );
165         gps_switch->setText("GPS");
166     }
167     else{
168         gps_switch->setStyleSheet(
169             "background-color: darkgray; "
170             "color: black; "
```

```
            //"font-weight: bold; "
            "border-radius: 5px; "
            "padding: 7px;"
            );
        gps_switch->setText("GPS");
    }


    status_egi->setFixedWidth(90);
    status_gps->setFixedWidth(90);
    status_egi->setAlignment(Qt::AlignCenter);
    status_gps->setAlignment(Qt::AlignCenter);

    status_egi->setFixedHeight(20);
    status_gps->setFixedHeight(20);
    egi_switch->setFixedWidth(50);
    gps_switch->setFixedWidth(50);

    if(isEgiRead){
        status_egi->setStyleSheet(
            "background-color: rgba(124, 158, 31, 0.9); "
            "color: gold; "
            "font-weight: bold; "
            );
    }
    else{
        status_egi->setStyleSheet(
            "background-color: rgba(234, 22, 22, 0.9); "
            "color: gold; "
            "font-weight: bold; "
            );
    }

    if(isGpsRead){
        status_gps->setStyleSheet(
            "background-color: rgba(124, 158, 31, 0.9); "
            "color: gold; "
            "font-weight: bold; "
            );
    }
    else{
        status_gps->setStyleSheet(
            "background-color: rgba(234, 22, 22, 0.9); "
            "color: gold; "
            "font-weight: bold; "
            );
    }


    f1->setFrameStyle(QFrame::Box | QFrame::Raised);
    f1->setLineWidth(2);
    f1->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);

    f2->setFrameStyle(QFrame::Box | QFrame::Raised);
    f2->setLineWidth(2);
    f2->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);

    QLabel *angle_label_s3 = new QLabel("°", this);
    QLabel *speed_label_s1 = new QLabel("feet/s", this);
    QLabel *speed_label_s2 = new QLabel("feet/s", this);
    QLabel *speed_label_s3 = new QLabel("feet/s", this);
    QLabel *acc_label_s1 = new QLabel("feet/s^2", this);
    QLabel *acc_label_s2 = new QLabel("feet/s^2", this);
    QLabel *acc_label_s3 = new QLabel("feet/s^2", this);
    QLabel *ac_speed_label_s = new QLabel("m/s", this);

    QLabel *lat_l_s = new QLabel("°", this);
    QLabel *lon_l_s = new QLabel("°", this);
    QLabel *alt_l_s = new QLabel("ft", this);

    QWidget *pan1;
    QWidget *pan2;
    QWidget *pan3;
    QWidget *pan4;
    QWidget *pan5;
```

```
246     QWidget *pan6;
247
248
249     QGridLayout *line1;
250     QGridLayout *line2;
251     QGridLayout *line3;
252     QHBoxLayout *line4;
253     QVBoxLayout *line5;
254     QHBoxLayout *line6;
255
256
257     map = new MapPanel(this);
258
259     f1->setLayout(new QVBoxLayout(this));
260
261     pan1 = new QWidget(this);
262     pan2 = new QWidget(this);
263     pan3 = new QWidget(this);
264     pan4 = new QWidget(this);
265     pan5 = new QWidget(this);
266     pan6 = new QWidget(this);
267
268     line1 = new QGridLayout(pan1);
269     line2 = new QGridLayout(pan2);
270     line3 = new QGridLayout(pan3);
271     line4 = new QHBoxLayout(pan4);
272     line5 = new QVBoxLayout(pan5);
273     line6 = new QHBoxLayout(pan6);
274
275
276     connect(egi_switch, &QPushButton::clicked, this, &PositionWidget::
        egi_switch_clicked);
277     connect(gps_switch, &QPushButton::clicked, this, &PositionWidget::
        gps_switch_clicked);
278
279     btn_DF = new QPushButton("DF", this);
280     btn_EGI = new QPushButton("EGI", this);
281     btn_GPS = new QPushButton("GPS", this);
282
283     btn_DF->setFixedSize(50, 25);
284     btn_EGI->setFixedSize(50, 25);
285     btn_GPS->setFixedSize(50, 25);
286
287
288     connect(btn_DF, &QPushButton::clicked, this, &PositionWidget::send_DF_kill
        );
289     connect(btn_GPS, &QPushButton::clicked, this, &PositionWidget::
        send_GPS_kill);
290     connect(btn_EGI, &QPushButton::clicked, this, &PositionWidget::
        send_EGI_kill);
291
292     connect(btn_DF, &QPushButton::pressed, this, &PositionWidget::switch_DF_HM
        );
293     connect(btn_GPS, &QPushButton::pressed, this, &PositionWidget::
        switch_GPS_HM);
294     connect(btn_EGI, &QPushButton::pressed, this, &PositionWidget::
        switch_EGI_HM);
295
296     connect(btn_DF, &QPushButton::released, this, &PositionWidget::
        switch_DF_HM);
297     connect(btn_GPS, &QPushButton::released, this, &PositionWidget::
        switch_GPS_HM);
298     connect(btn_EGI, &QPushButton::released, this, &PositionWidget::
        switch_EGI_HM);
299
300
301     //BUTTON colours for APPS
302
303     //DF
304
305     if(!DF_APP_pressed && DF_APP){ //not pressed & app ok
306         btn_DF->setStyleSheet(
307             "background-color: green; "
308             "color: gold; "
309             "font-weight: bold; "
```

```
310             "border-radius: 5px; "
311             "padding: 7px;"
312             );
313     }
314     else if(DF_APP_pressed && DF_APP){ //pressed & app ok
315         btn_DF->setStyleSheet(
316             "background-color: darkgreen; "
317             "color: gold; "
318             "font-weight: bold; "
319             "border-radius: 5px; "
320             "padding: 7px;"
321             );
322     }
323     else if(!DF_APP_pressed && !DF_APP){ //not pressed & app not ok
324         btn_DF->setStyleSheet(
325             "background-color: red; "
326             "color: gold; "
327             "font-weight: bold; "
328             "border-radius: 5px; "
329             "padding: 7px;"
330             );
331     }
332     else{ // DF_APP_pressed && !DF_APP //pressed & app not ok
333         btn_DF->setStyleSheet(
334             "background-color: darkred; "
335             "color: gold; "
336             "font-weight: bold; "
337             "border-radius: 5px; "
338             "padding: 7px;"
339             );
340     }
341
342     //GPS
343
344     if(!GPS_APP_pressed && GPS_APP){ //not pressed & app ok
345         btn_GPS->setStyleSheet(
346             "background-color: green; "
347             "color: gold; "
348             "font-weight: bold; "
349             "border-radius: 5px; "
350             "padding: 7px;"
351             );
352     }
353     else if(GPS_APP_pressed && GPS_APP){ //pressed & app ok
354         btn_GPS->setStyleSheet(
355             "background-color: darkgreen; "
356             "color: gold; "
357             "font-weight: bold; "
358             "border-radius: 5px; "
359             "padding: 7px;"
360             );
361     }
362     else if(!GPS_APP_pressed && !GPS_APP){ //not pressed & app not ok
363         btn_GPS->setStyleSheet(
364             "background-color: red; "
365             "color: gold; "
366             "font-weight: bold; "
367             "border-radius: 5px; "
368             "padding: 7px;"
369             );
370     }
371     else{ // GPS_APP_pressed && !GPS_APP //pressed & app not ok
372         btn_GPS->setStyleSheet(
373             "background-color: darkred; "
374             "color: gold; "
375             "font-weight: bold; "
376             "border-radius: 5px; "
377             "padding: 7px;"
378             );
379     }
380
381     //EGI
382
383     if(!EGI_APP_pressed && EGI_APP){ //not pressed & app ok
384         btn_EGI->setStyleSheet(
```

```
385             "background-color: green; "
386             "color: gold; "
387             "font-weight: bold; "
388             "border-radius: 5px; "
389             "padding: 7px;"
390             );
391     }
392     else if(EGI_APP_pressed && EGI_APP){ //pressed & app ok
393         btn_EGI->setStyleSheet(
394             "background-color: darkgreen; "
395             "color: gold; "
396             "font-weight: bold; "
397             "border-radius: 5px; "
398             "padding: 7px;"
399             );
400     }
401     else if(!EGI_APP_pressed && !EGI_APP){ //not pressed & app not ok
402         btn_EGI->setStyleSheet(
403             "background-color: red; "
404             "color: gold; "
405             "font-weight: bold; "
406             "border-radius: 5px; "
407             "padding: 7px;"
408             );
409     }
410     else{ // EGI_APP_pressed && !EGI_APP //pressed & app not ok
411         btn_EGI->setStyleSheet(
412             "background-color: darkred; "
413             "color: gold; "
414             "font-weight: bold; "
415             "border-radius: 5px; "
416             "padding: 7px;"
417             );
418     }
419
420
421     // COORDS
422     line1->addWidget(lat_l, 0, 0, 1, 1);
423     line1->addWidget(lat_t, 0, 1, 1, 1);
424     line1->addWidget(lat_l_s, 0, 2, 1, 1);
425
426     line1->addWidget(lon_l, 1, 0, 1, 1);
427     line1->addWidget(lon_t, 1, 1, 1, 1);
428     line1->addWidget(lon_l_s, 1, 2, 1, 1);
429
430     line1->addWidget(alt_l, 2, 0, 1, 1);
431     line1->addWidget(alt_t, 2, 1, 1, 1);
432     line1->addWidget(alt_l_s, 2, 2, 1, 1);
433
434
435     //SPEED
436     line1->addWidget(speed_x_l, 0, 3, 1, 1);
437     line1->addWidget(speed_x, 0, 4, 1, 1);
438     line1->addWidget(speed_label_s1, 0, 5, 1, 1);
439
440     line1->addWidget(speed_y_l, 1, 3, 1, 1);
441     line1->addWidget(speed_y, 1, 4, 1, 1);
442     line1->addWidget(speed_label_s2, 1, 5, 1, 1);
443
444     line1->addWidget(speed_z_l, 2, 3, 1, 1);
445     line1->addWidget(speed_z, 2, 4, 1, 1);
446     line1->addWidget(speed_label_s3, 2, 5, 1, 1);
447
448     //ACC
449     line1->addWidget(acc_x_l, 0, 6, 1, 1);
450     line1->addWidget(acc_x, 0, 7, 1, 1);
451     line1->addWidget(acc_label_s1, 0, 8, 1, 1);
452
453     line1->addWidget(acc_y_l, 1, 6, 1, 1);
454     line1->addWidget(acc_y, 1, 7, 1, 1);
455     line1->addWidget(acc_label_s2, 1, 8, 1, 1);
456
457     line1->addWidget(acc_z_l, 2, 6, 1, 1);
458     line1->addWidget(acc_z, 2, 7, 1, 1);
459     line1->addWidget(acc_label_s3, 2, 8, 1, 1);
```

```
460
461     //HEADING
462     line2->addWidget(head_l, 0, 0, 1, 1);
463     line2->addWidget(head_t, 0, 1, 1, 1);
464     line2->addWidget(angle_label_s3, 0, 2, 1, 1);
465
466     //AC_SPEED
467     line2->addWidget(ac_speed_l, 0, 3, 1, 1);
468     line2->addWidget(ac_speed_t, 0, 4, 1, 1);
469     line2->addWidget(ac_speed_label_s, 0, 5, 1, 1);
470
471
472     // status flag
473     line3->addWidget(status_egi, 0, 0, 1, 1);
474     line3->addWidget(btn_EGI, 0, 1, 1, 1);
475
476     line3->addWidget(status_gps, 1, 0, 1, 1);
477     line3->addWidget(btn_GPS, 1, 1, 1, 1);
478
479     line3->addWidget(btn_DF, 0, 2, 1, 1);
480
481
482     //status of apps
483     line4->addWidget(pan2);
484
485     f1->layout()->addWidget(data_frame_label);
486     f1->layout()->addWidget(pan1);
487     f1->layout()->addWidget(pan4);
488     f1->layout()->addWidget(pan3);
489     f1->layout()->addWidget(pan6);
490
491     f2->setLayout(new QVBoxLayout(this));
492
493     QWidget *panel_label_btn = new QWidget(this);
494     QHBoxLayout *label_btn_ly = new QHBoxLayout(panel_label_btn);
495     panel_label_btn->setFixedHeight(50);
496
497     //status activation button
498     label_btn_ly->addWidget(map_frame_label);
499     label_btn_ly->addWidget(egi_switch);
500     label_btn_ly->addWidget(gps_switch);
501
502     f2->layout()->addWidget(panel_label_btn);
503     f2->layout()->addWidget(map);
504
505     layout->addWidget(f1);
506     layout->addWidget(f2);
507
508     setLayout(layout);
509 }
510
511 void PositionWidget::paintEvent(QPaintEvent *event)
512 {
513     QPainter painter(this);
514     painter.setRenderHint(QPainter::Antialiasing);
515
516 }
517
518 void PositionWidget::updatePosition()
519 {
520
521     //kill_msg.option = 0;
522
523     if(isEgiOn){
524         egi_switch->setStyleSheet(
525             "background-color: lightgrey; "
526             "color: black; "
527             //"font-weight: bold; "
528             "border-radius: 5px; "
529             "padding: 7px;"
530             );
531         egi_switch->setText("EGI");
532         this->map->setIsEgiOn(true);
533     }
534     else{
```

```
535         egi_switch->setStyleSheet(
536             "background-color: darkgrey; "
537             "color: black; "
538             //"font-weight: bold; "
539             "border-radius: 5px; "
540             "padding: 7px;"
541             );
542         egi_switch->setText("EGI");
543         this->map->setIsEgiOn(false);
544     }
545 
546     if(isGpsOn){
547         gps_switch->setStyleSheet(
548             "background-color: lightgrey; "
549             "color: black; "
550             //"font-weight: bold; "
551             "border-radius: 5px; "
552             "padding: 7px;"
553             );
554         gps_switch->setText("GPS");
555         this->map->setIsGpsOn(true);
556     }
557     else{
558         gps_switch->setStyleSheet(
559             "background-color: darkgrey; "
560             "color: black; "
561             //"font-weight: bold; "
562             "border-radius: 5px; "
563             "padding: 7px;"
564             );
565         gps_switch->setText("GPS");
566         this->map->setIsGpsOn(false);
567     }
568 
569 
570     if(isEgiRead){
571         status_egi->setStyleSheet(
572             "background-color: rgba(124, 158, 31, 0.9); "
573             "color: gold; "
574             "font-weight: bold; "
575             );
576         this->map->setIsEgiRead(true);
577     }
578     else{
579         status_egi->setStyleSheet(
580             "background-color: rgba(234, 22, 22, 0.9); "
581             "color: gold; "
582             "font-weight: bold; "
583             );
584         this->map->setIsEgiRead(false);
585     }
586 
587     if(isGpsRead){
588         status_gps->setStyleSheet(
589             "background-color: rgba(124, 158, 31, 0.9); "
590             "color: gold; "
591             "font-weight: bold; "
592             );
593         this->map->setIsGpsRead(true);
594     }
595     else{
596         status_gps->setStyleSheet(
597             "background-color: rgba(234, 22, 22, 0.9); "
598             "color: gold; "
599             "font-weight: bold; "
600             );
601         this->map->setIsGpsRead(false);
602     }
603 
604     this->map->setIsPlane();
605 
606     this->map->setPosition(lat, lon, alt, heading, yaw);
607     this->map->setPosition(lat_egi, lon_egi, 0);
608     this->map->setPosition(lat_gps, lon_gps, 1);
609     this->map->update();
```

```
610
611     this->head_t->setText(QString::number((short)rad2deg(heading)));
612     this->lat_t->setText(double2dms_string(lat));
613     this->lon_t->setText(double2dms_string(lon));
614     this->alt_t->setText(QString::number(alt));
615
616     if(isEgiRead){
617         this->speed_x->setText(QString::number(speed.X, 'f', 3));
618         this->speed_y->setText(QString::number(speed.Y, 'f', 3));
619         this->speed_z->setText(QString::number(speed.Z, 'f', 3));
620
621         this->acc_x->setText(QString::number(acc.X, 'f', 2));
622         this->acc_y->setText(QString::number(acc.Y, 'f', 2));
623         this->acc_z->setText(QString::number(acc.Z, 'f', 2));
624     }
625     else{
626         this->speed_x->setText("------");
627         this->speed_y->setText("------");
628         this->speed_z->setText("------");
629
630         this->acc_x->setText("------");
631         this->acc_y->setText("------");
632         this->acc_z->setText("------");
633     }
634
635
636     this->ac_speed_t->setText(QString::number(ac_speed));
637
638     //DF
639
640     if(!DF_APP_pressed && DF_APP){ //not pressed & app ok
641         btn_DF->setStyleSheet(
642             "background-color: green; "
643             "color: gold; "
644             "font-weight: bold; "
645             "border-radius: 5px; "
646             "padding: 7px;"
647             );
648     }
649     else if(DF_APP_pressed && DF_APP){ //pressed & app ok
650         btn_DF->setStyleSheet(
651             "background-color: darkgreen; "
652             "color: gold; "
653             "font-weight: bold; "
654             "border-radius: 5px; "
655             "padding: 7px;"
656             );
657     }
658     else if(!DF_APP_pressed && !DF_APP){ //not pressed & app not ok
659         btn_DF->setStyleSheet(
660             "background-color: red; "
661             "color: gold; "
662             "font-weight: bold; "
663             "border-radius: 5px; "
664             "padding: 7px;"
665             );
666     }
667     else{ // DF_APP_pressed && !DF_APP //pressed & app not ok
668         btn_DF->setStyleSheet(
669             "background-color: darkred; "
670             "color: gold; "
671             "font-weight: bold; "
672             "border-radius: 5px; "
673             "padding: 7px;"
674             );
675     }
676
677     //GPS
678
679     if(!GPS_APP_pressed && GPS_APP){ //not pressed & app ok
680         btn_GPS->setStyleSheet(
681             "background-color: green; "
682             "color: gold; "
683             "font-weight: bold; "
684             "border-radius: 5px; "
```

```
685             "padding: 7px;"
686             );
687     }
688     else if(GPS_APP_pressed && GPS_APP){ //pressed & app ok
689         btn_GPS->setStyleSheet(
690             "background-color: darkgreen; "
691             "color: gold; "
692             "font-weight: bold; "
693             "border-radius: 5px; "
694             "padding: 7px;"
695             );
696     }
697     else if(!GPS_APP_pressed && !GPS_APP){ //not pressed & app not ok
698         btn_GPS->setStyleSheet(
699             "background-color: red; "
700             "color: gold; "
701             "font-weight: bold; "
702             "border-radius: 5px; "
703             "padding: 7px;"
704             );
705     }
706     else{ // GPS_APP_pressed && !GPS_APP //pressed & app not ok
707         btn_GPS->setStyleSheet(
708             "background-color: darkred; "
709             "color: gold; "
710             "font-weight: bold; "
711             "border-radius: 5px; "
712             "padding: 7px;"
713             );
714     }
715
716     //EGI
717
718     if(!EGI_APP_pressed && EGI_APP){ //not pressed & app ok
719         btn_EGI->setStyleSheet(
720             "background-color: green; "
721             "color: gold; "
722             "font-weight: bold; "
723             "border-radius: 5px; "
724             "padding: 7px;"
725             );
726     }
727     else if(EGI_APP_pressed && EGI_APP){ //pressed & app ok
728         btn_EGI->setStyleSheet(
729             "background-color: darkgreen; "
730             "color: gold; "
731             "font-weight: bold; "
732             "border-radius: 5px; "
733             "padding: 7px;"
734             );
735     }
736     else if(!EGI_APP_pressed && !EGI_APP){ //not pressed & app not ok
737         btn_EGI->setStyleSheet(
738             "background-color: red; "
739             "color: gold; "
740             "font-weight: bold; "
741             "border-radius: 5px; "
742             "padding: 7px;"
743             );
744     }
745     else{ // EGI_APP_pressed && !EGI_APP //pressed & app not ok
746         btn_EGI->setStyleSheet(
747             "background-color: darkred; "
748             "color: gold; "
749             "font-weight: bold; "
750             "border-radius: 5px; "
751             "padding: 7px;"
752             );
753     }
754
755     update();
756 }
757
758
759 void PositionWidget::load_airplane(){
```

```
760
761     FILE *fp = fopen("FACE_LAYERS/PSSS/AC_POS_APP_GRAPHIC/plane_model.csv", "r
        ");
762
763     if (fp == NULL) {
764         printf("Error during file opening: %s\n", strerror(errno));
765         exit(1);
766     }
767
768     char riga[65];
769
770     int i=0,j;
771
772     while(fscanf(fp, "%s", riga)!=EOF){
773         int k = 0;
774         for(j=0; j<strlen(riga); j++){
775             if(riga[j]!=';'){
776                 if(riga[j]=='0'){
777                     plane[i][k++] = 0;
778                 }
779                 if(riga[j]=='1'){
780                     plane[i][k++] = 1;
781                 }
782             }
783         }
784         i++;
785     }
786
787     fclose(fp);
788 }
789
790 void PositionWidget::load_rose(){
791
792     FILE *fp = fopen("rosa_dei_venti.csv", "r");
793
794     if (fp == NULL) {
795         printf("Error during file opening: %s\n", strerror(errno));
796         exit(1);
797     }
798
799     char riga[150];
800
801     int i=0,j;
802
803     while(fscanf(fp, "%s", riga)!=EOF){
804         int k = 0;
805         for(j=0; j<strlen(riga); j++){
806             if(riga[j]!=';'){
807                 if(riga[j]=='0'){
808                     rose[i][k++] = 0;
809                 }
810                 if(riga[j]=='1'){
811                     rose[i][k++] = 1;
812                 }
813             }
814         }
815         i++;
816     }
817
818     fclose(fp);
819 }
820
821 double PositionWidget::get_lat() const {
822     return this->lat;
823 }
824
825 double PositionWidget::get_lon() const {
826     return this->lon;
827 }
828
829 int PositionWidget::get_alt() const {
830     return this->alt;
831 }
832
833 float PositionWidget::get_heading() const {
```

```
834     return this->heading;
835 }
836
837 // Implementazione dei setter
838 void PositionWidget::set_lat(double lat) {
839     this->lat = lat;
840 }
841
842 void PositionWidget::set_lon(double lon) {
843     this->lon = lon;
844 }
845
846 void PositionWidget::set_lat(double lat, int opt) {
847     switch(opt){
848         case 0:{
849             this->lat_egi = lat;
850             break;
851         }
852         case 1:{
853             this->lat_gps = lat;
854             break;
855         }
856     }
857 }
858
859 void PositionWidget::set_lon(double lon, int opt) {
860     switch(opt){
861         case 0:{
862             this->lon_egi = lon;
863             break;
864         }
865         case 1:{
866             this->lon_gps = lon;
867             break;
868         }
869     }
870 }
871
872 void PositionWidget::set_alt(int alt) {
873     this->alt = alt;
874 }
875
876 void PositionWidget::set_heading(float heading) {
877     this->heading = heading;
878 }
879
880 QString PositionWidget::double2dms_string(const double value){
881     Coordinate_t c = decToDMS_r(value);
882     QString ris =  QString("%1° %2' %3''")
883                        .arg(c.degree)
884                        .arg(c.min)
885                        .arg((int) c.sec);
886     return ris;
887 }
888
889 double PositionWidget::get_ac_speed() const
890 {
891     return this->ac_speed;
892 }
893
894 void PositionWidget::set_ac_speed(double ac_speed)
895 {
896     this->ac_speed = ac_speed;
897 }
898
899 speed_t PositionWidget::get_speed() const
900 {
901     return this->speed;
902 }
903
904 void PositionWidget::set_speed(const speed_t speed)
905 {
906     this->speed = speed;
907 }
908
```

```
909 acc_t PositionWidget::get_acc() const
910 {
911     return this->acc;
912 }
913
914 void PositionWidget::set_acc(const acc_t acc)
915 {
916     this->acc = acc;
917 }
918
919 void PositionWidget::setEgiRead(bool val){
920     this->isEgiRead = val;
921     this->map->setIsEgiRead(val);
922     this->map->setIsPlane();
923 }
924 bool PositionWidget::getEgiRead(){
925     return this->isEgiRead;
926 }
927 void PositionWidget::setGpsRead(bool val){
928     this->isGpsRead = val;
929     this->map->setIsGpsRead(val);
930     this->map->setIsPlane();
931 }
932 bool PositionWidget::getGpsRead(){
933     return this->isGpsRead;
934 }
935
936 void PositionWidget::setEgiOn(bool val){
937     this->isEgiOn = val;
938     this->map->setIsEgiOn(val);
939     this->map->setIsPlane();
940 }
941 bool PositionWidget::getEgiOn(){
942     return this->isEgiOn;
943 }
944 void PositionWidget::setGpsOn(bool val){
945     this->isGpsOn = val;
946     this->map->setIsGpsOn(val);
947     this->map->setIsPlane();
948 }
949 bool PositionWidget::getGpsOn(){
950     return this->isGpsOn;
951 }
952
953 void PositionWidget::set_yaw(short y){
954     this->yaw = y;
955 }
956
957 short PositionWidget::get_yaw(){
958     return this->yaw;
959 }
960
961 void PositionWidget::egi_switch_clicked(){
962     this->isEgiOn = !this->isEgiOn;
963     setEgiOn(this->isEgiOn);
964
965 }
966 void PositionWidget::gps_switch_clicked(){
967     this->isGpsOn = !this->isGpsOn;
968     setGpsOn(this->isGpsOn);
969 }
970
971 void PositionWidget::set_EGI_APP(bool v){
972     this->EGI_APP = v;
973 }
974 void PositionWidget::set_GPS_APP(bool v){
975     this->GPS_APP = v;
976 }
977 void PositionWidget::set_DF_APP(bool v){
978     this->DF_APP = v;
979 }
980
981 bool PositionWidget::get_EGI_APP(){
982     return this->EGI_APP;
983 }
```

```
984 bool PositionWidget::get_GPS_APP(){
985     return this->GPS_APP;
986 }
987 bool PositionWidget::get_DF_APP(){
988     return this->DF_APP;
989 }
990
991 void PositionWidget::switch_DF_HM(){
992     this->DF_APP_pressed = !this->DF_APP_pressed;
993 }
994 void PositionWidget::switch_GPS_HM(){
995     this->GPS_APP_pressed = !this->GPS_APP_pressed;
996 }
997 void PositionWidget::switch_EGI_HM(){
998     this->EGI_APP_pressed = !this->EGI_APP_pressed;
999 }
1000
1001
1002 void PositionWidget::send_DF_kill(){
1003 #ifdef USE_TSS
1004     //printf("Killing DF...\n");
1005     kill_msg.option = 1; //DF
1006     memcpy((void*) wr.payload, (void*) &kill_msg, sizeof(FACE::HM_KILL_t));
1007     ser_hm.Write(handle_hm, FACE::NO_WAIT, &wr, sizeof(FACE::HM_KILL_t),
        status);
1008     //send_kill_msg(socket_hm, kill_msg, status);
1009     kill_msg.option = 0;
1010 #endif
1011 }
1012
1013 void PositionWidget::send_EGI_kill(){
1014 #ifdef USE_TSS
1015     //printf("Killing EGI...\n");
1016     kill_msg.option = 2; //EGI
1017     memcpy((void*) wr.payload, (void*) &kill_msg, sizeof(FACE::HM_KILL_t));
1018     ser_hm.Write(handle_hm, FACE::NO_WAIT, &wr, sizeof(FACE::HM_KILL_t),
        status);
1019     //send_kill_msg(socket_hm, kill_msg, status);
1020     kill_msg.option = 0;
1021 #endif
1022 }
1023
1024 void PositionWidget::send_GPS_kill(){
1025 #ifdef USE_TSS
1026     //printf("Killing GPS...\n");
1027     kill_msg.option = 3; //GPS
1028     memcpy((void*) wr.payload, (void*) &kill_msg, sizeof(FACE::HM_KILL_t));
1029     ser_hm.Write(handle_hm, FACE::NO_WAIT, &wr, sizeof(FACE::HM_KILL_t),
        status);
1030     //send_kill_msg(socket_hm, kill_msg, status);
1031     kill_msg.option = 0;
1032 #endif
1033 }
1034
1035 // wrapper for serial.Read to be called out of this section
1036 void PositionWidget::read_serial(char *buf, FACE::RETURN_CODE_TYPE &status){
1037 #ifdef USE_TSS
1038     ser_hm.Read(handle_hm, FACE::NO_WAIT, &rd, sizeof(FACE::HM_APP_t), status)
        ;
1039     buf = (char*) rd.payload;
1040 #endif
1041 }
1042
1043
1044 // useless, only for completeness purpose
1045 void PositionWidget::write_serial(char *buf, FACE::RETURN_CODE_TYPE &status){
1046 #ifdef USE_TSS
1047     wr.payload = (FACE::IOSS::Serial::PAYLOAD_DATA_MSG_TYPE) buf;
1048     ser_hm.Write(handle_hm, FACE::NO_WAIT, &wr, sizeof(FACE::HM_KILL_t),
        status);
1049 #endif
1050 }
1051
1052 PositionWidget::~PositionWidget(){
1053 #ifdef USE_TSS
```

```
1054     ser_hm.Flush(handle_hm, status);
1055     ser_hm.Close_Connection(handle_hm, status);
1056 #endif
1057 }
```

A.3.5.3 map_panel.h

```
1  #ifndef MAPPANEL_H
2  #define MAPPANEL_H
3
4  #include <QWidget>
5  #include <QPainter>
6  #include <math.h>
7
8  class MapPanel : public QWidget {
9      Q_OBJECT
10 public:
11     explicit MapPanel(QWidget *parent = nullptr);
12     void setPlane(int plane[35][33]);
13     void setRose(int rose[66][60]);
14     void setPosition(double lat, double lon, int alt, float heading, short yaw
       );
15     void setPosition(double lat, double lon, int opz);
16     void setIsEgiRead(bool v);
17     void setIsGpsRead(bool v);
18     void setIsEgiOn(bool v);
19     void setIsGpsOn(bool v);
20     void setIsPlane(bool v);
21     void setIsPlane();
22
23 protected:
24     void paintEvent(QPaintEvent *event) override;
25
26 private:
27
28     double center_lat = 45.0;
29     double center_lon = 7.6;
30
31     int plane[35][33];
32     int rose[66][60];
33     int Cx = 17;
34     int Cy = 16;
35     int lon;
36     int lat;
37
38     int lat_egi;
39     int lon_egi;
40     int lat_gps;
41     int lon_gps;
42
43     float heading;
44     short yaw;
45     int alt;
46     int center_X;
47     int center_Y;
48
49     bool isPlane;
50     bool isEgiRead;
51     bool isGpsRead;
52
53     bool isEgiOn;
54     bool isGpsOn;
55 };
56
57 #endif // MAPPANEL_H
```

A.3.5.4 map_panel.cpp

```
#include "map_panel.h"

const double max_lat_dp = 1.2;
const double max_lon_dp = 1.2;


MapPanel::MapPanel(QWidget *parent) : QWidget(parent) {
    this->center_X = this->height()/8;
    this->center_Y = this->width()*1.5;
    this->isPlane = true;
    this->isEgiRead = true;
    this->isGpsRead = true;
    this->isEgiOn = true;
    this->isGpsOn = true;
}

void MapPanel::paintEvent(QPaintEvent *event) {
    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing);

    //painter.setBrush(QColor(155, 118, 83));
    painter.setBrush(Qt::green);
    painter.setOpacity(0.2);
    painter.drawRect(0, 0, this->width(), this->height());

    painter.setOpacity(1.0);
    painter.setBrush(Qt::black);
    painter.drawText(12, 20, "EGI");
    painter.drawText(12, 40, "GPS");

    painter.setBrush(Qt::blue); //EGI
    painter.drawEllipse(3, 12, 5, 5);
    painter.setBrush(Qt::red);  //RED
    painter.drawEllipse(3, 32, 5, 5);


    if(isEgiRead && isEgiOn){
        painter.setBrush(Qt::blue);
        painter.drawEllipse(Cx+lon_egi-2, Cy+lat_egi-2, 5, 5);
        //printf("%d\t", isEgi);
        fflush(stdout);
    }

    if(isGpsRead && isGpsOn){
        painter.setBrush(Qt::red);
        painter.drawEllipse(Cx+lon_gps-2, Cy+lat_gps-2, 5, 5);
        //printf("%d\t", isGps);
        fflush(stdout);
    }

    if(isPlane){
        //printf("%d\n", isPlane);
        fflush(stdout);

        painter.setBrush(Qt::darkGray);
        painter.setOpacity(0.4);
        double M[2][2];
        double head_d = (double)heading + (double)yaw*M_PI/180.0;

        M[0][0] = cos(head_d);
        M[1][1] = cos(head_d);
        M[0][1] = -sin(head_d);
        M[1][0] = sin(head_d);

        int i,j;
        for(i=0; i<35; i++){
            for(j=0; j<33; j++){
                if(plane[i][j]==1){

                    //centered coordinates
                    int y = i-Cy;
                    int x = j-Cx;

```

```
74                      //rotated point
75                      double x_n = M[0][0]*(double)x + M[0][1]*(double)y;
76                      double y_n = M[1][0]*(double)x + M[1][1]*(double)y;
77
78                      // back to orginal coordinates
79                      x = (int)x_n+Cx+this->lon;
80                      y = (int)y_n+Cy+this->lat;
81
82                      painter.drawPoint(x,y);
83                  }
84              }
85          }
86      }
87
88      //draw ROSE
89      painter.setOpacity(0.8);
90      painter.setBrush(Qt::black);
91
92      int i,j;
93
94      int h = this->size().height();
95      int w = this->size().width();
96
97      for(i=0; i<66; i++){
98          for(j=0; j<60; j++){
99              if(rose[i][j]==1){
100
101                 int y = i+(h-80);
102                 int x = j+(w-80);
103
104                 painter.drawPoint(x,y);
105             }
106         }
107     }
108
109 }
110
111
112 void MapPanel::setPlane(int plane[35][33]){
113     for(int i=0; i<35; i++){
114         for(int j=0; j<33; j++){
115             this->plane[i][j] = plane[i][j];
116         }
117     }
118 }
119
120 void MapPanel::setRose(int rose[66][60]){
121     for(int i=0; i<66; i++){
122         for(int j=0; j<60; j++){
123             this->rose[i][j] = rose[i][j];
124         }
125     }
126 }
127
128
129 // plane
130 void MapPanel::setPosition(double lat, double lon, int alt, float heading,
        short yaw){
131
132     //printf("%f %f %f %f\n", lat, lon, center_lat, center_lon);
133
134
135     if(lat>(center_lat+max_lat_dp/2.0)){
136         center_lat = center_lat+max_lat_dp;
137     }
138     if(lat<(center_lat-max_lat_dp/2.0)){
139         center_lat = center_lat-max_lat_dp;
140     }
141
142     if(lon>(center_lon+max_lon_dp/2.0)){
143         center_lon = center_lon+max_lon_dp;
144     }
145     if(lon<(center_lon-max_lon_dp/2.0)){
146         center_lon = center_lon-max_lon_dp;
147     }
```

```
148
149     double cvX = max_lon_dp/(double)this->width();
150     double cvY = max_lat_dp/(double)this->height();
151
152     this->heading = heading;
153     this->lon = this->center_X+(int)((lon-center_lon)/cvX);
154     this->lat = this->center_Y-(int)((lat-center_lat)/cvY);
155     this->alt = alt;
156     this->yaw = yaw;
157
158 }
159
160
161 //EGI & GPS
162 void MapPanel::setPosition(double lat, double lon, int opz){
163
164     double cvX = max_lon_dp/(double)this->width();
165     double cvY = max_lat_dp/(double)this->height();
166
167     switch(opz){
168         case 0:{
169             this->lon_egi = this->center_X+(int)((lon-center_lon)/cvX);
170             this->lat_egi = this->center_Y-(int)((lat-center_lat)/cvY);
171             break;
172         }
173         case 1:{
174             this->lon_gps = this->center_X+(int)((lon-center_lon)/cvX);
175             this->lat_gps = this->center_Y-(int)((lat-center_lat)/cvY);
176             break;
177         }
178     }
179 }
180
181 void MapPanel::setIsEgiRead(bool v){
182     this->isEgiRead = v;
183 }
184 void MapPanel::setIsGpsRead(bool v){
185     this->isGpsRead = v;
186 }
187
188 void MapPanel::setIsEgiOn(bool v){
189     this->isEgiOn = v;
190 }
191 void MapPanel::setIsGpsOn(bool v){
192     this->isGpsOn = v;
193 }
194
195 void MapPanel::setIsPlane(bool v){
196     this->isPlane = v;
197 }
198 void MapPanel::setIsPlane(){
199     this->isPlane = this->isEgiRead && this->isGpsRead && this->isEgiOn &&
    this->isGpsOn;
200 }
```

A.3.5.5 right_screen.h

```
1  #ifndef CUSTOMWIDGET_RIGHT_H
2  #define CUSTOMWIDGET_RIGHT_H
3
4  #include <QWidget>
5  #include <QTimer>
6  #include <QPainter>
7  #include <QLabel>
8  #include <QSplitter>
9  #include <QVBoxLayout>
10 #include <QMenu>
11 #include <QMenuBar>
12 #include <QPainterPath>
13 #include <math.h>
14
```

```
#define MAX_ROLL 30
#define MAX_PITCH 30

class AttitudeWidget : public QWidget
{
    Q_OBJECT

public:
    explicit AttitudeWidget(QWidget *parent = nullptr);

    void set_roll(short roll);
    void set_pitch(short pitch);
    void set_yaw(short yaw);
    short get_roll() const;
    short get_pitch() const;
    short get_yaw() const;

protected:
    void paintEvent(QPaintEvent *event) override;

public slots:
    void updateAttitude();

private:
    void setupUI();
    //QTimer *timer;
    short roll;
    short pitch;
    short yaw;

    //short offset;

    //QWidget *w;

};



#endif // CUSTOMWIDGET_RIGHT_H
```

A.3.5.6 right_screen.cpp

```

#include "right_screen.h"
#define BUFFER_SIZE 4

AttitudeWidget::AttitudeWidget(QWidget *parent) : QWidget(parent) {

    //offset = 30;

    //timer = new QTimer(this);
    //connect(timer, &QTimer::timeout, this, &AttitudeWidget::updateAttitude);
    //timer->start(100);

}

void AttitudeWidget::paintEvent(QPaintEvent *event) {
    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing);

    QSize s = this->size();

    // Definisci il centro e il raggio del cerchio dell'orizzonte artificiale
    int radius = s.height() / 4; // Ad esempio, un quarto dell'altezza del
    widget
    int center_x = s.width() / 2;
    int center_y = s.height() / 2;

    double roll_d = (double)roll * M_PI / 180.0;
    double pitch_d = (double)pitch * M_PI / 180.0;
    double yaw_d = (double)yaw * M_PI / 180.0;
```

```
29
30      float slope = tan(roll_d); // Coefficiente angolare della retta
31      int yOffset = static_cast<int>(center_y + tan(pitch_d) * radius);
32
33      // Disegna il cerchio dell'orizzonte artificiale
34      painter.setPen(QPen(Qt::black, 2));
35
36      // Disegna il cielo e il terreno all'interno del cerchio
37      QPainterPath skyPath;
38      QPainterPath groundPath;
39
40      int y1 = yOffset + slope * (-radius);
41      int y2 = yOffset + slope * radius;
42
43      if(y2<center_y-radius){  // OK
44          skyPath.moveTo(center_x - radius, center_y - radius);
45          skyPath.lineTo(center_x - (int)((float)(y1-(center_y-radius))/slope) -
     radius, center_y - radius);
46          skyPath.lineTo(center_x - radius, y1);
47          skyPath.lineTo(center_x - radius, center_y - radius);
48
49
50          groundPath.moveTo(center_x - (int)((float)(y1-(center_y-radius))/slope
    ) - radius, center_y - radius); //1
51          groundPath.lineTo(center_x + radius, center_y - radius); //2
52          groundPath.lineTo(center_x + radius, center_y + radius); //3
53          groundPath.lineTo(center_x - radius, center_y + radius); //4
54          groundPath.lineTo(center_x - radius, y1); //5
55          groundPath.lineTo(center_x - (int)((float)(y1-(center_y-radius))/slope
    ) - radius, center_y - radius); //1
56      }
57      else if(y1<center_y-radius){ // OK
58          skyPath.moveTo(center_x - (int)((float)(y2-(center_y-radius))/slope) +
     radius, center_y - radius);
59          skyPath.lineTo(center_x + radius, center_y - radius); // 2
60          skyPath.lineTo(center_x + radius, y2); // 3
61          skyPath.lineTo(center_x - (int)((float)(y2-(center_y-radius))/slope) +
     radius, center_y - radius);
62
63          groundPath.moveTo(center_x - radius, center_y - radius); // 1
64          groundPath.lineTo(center_x - (int)((float)(y2-(center_y-radius))/slope
    ) + radius, center_y - radius); // 2
65          groundPath.lineTo(center_x + radius, y2); // 3
66          groundPath.lineTo(center_x + radius, center_y + radius); // 4
67          groundPath.lineTo(center_x - radius, center_y + radius); // 5
68          groundPath.lineTo(center_x - radius, center_y - radius); // 1
69      }
70
71
72      //if(pitch>0){ // pitch positive (DOWN)
73      else if(y2>center_y+radius){ //OK
74          skyPath.moveTo(center_x - radius, center_y - radius); //1
75          skyPath.lineTo(center_x + radius, center_y - radius); //2
76          skyPath.lineTo(center_x + radius, center_y + radius); //3
77          skyPath.lineTo(center_x + (int)((float)(center_y+radius-y1)/slope) -
    radius, center_y + radius); //4
78          skyPath.lineTo(center_x - radius, y1); //5
79          skyPath.lineTo(center_x - radius, center_y - radius); //1
80
81          groundPath.moveTo(center_x - radius, center_y + radius); //1
82          groundPath.lineTo(center_x + (int)((float)(center_y+radius-y1)/slope)
    - radius, center_y + radius); //2
83          groundPath.lineTo(center_x - radius, y1); //3
84          groundPath.lineTo(center_x - radius, center_y + radius); //1
85      }
86      else if(y1>center_y+radius){ // roll < 0 OK
87          skyPath.moveTo(center_x - radius, center_y - radius); //1
88          skyPath.lineTo(center_x + radius, center_y - radius); //2
89          skyPath.lineTo(center_x + radius, y2); //3
90          skyPath.lineTo(center_x + (int)((float)(center_y+radius-y2)/slope) +
    radius, center_y+radius); //4
91          skyPath.lineTo(center_x - radius, center_y + radius); //5
92          skyPath.lineTo(center_x - radius, center_y - radius); //1
93
94          groundPath.moveTo(center_x + radius, center_y + radius); //1
```

```
95          groundPath.lineTo(center_x + (int)((float)(center_y+radius-y2)/slope)
    + radius, center_y + radius); //2
96          groundPath.lineTo(center_x + radius, y2); //3
97          groundPath.lineTo(center_x + radius, center_y + radius); //1
98      }
99      //}
100     else{ // pitch OK
101         skyPath.moveTo(center_x - radius, center_y - radius);
102         skyPath.lineTo(center_x + radius, center_y - radius);
103         skyPath.lineTo(center_x + radius, y2);
104         skyPath.lineTo(center_x - radius, y1);
105         skyPath.lineTo(center_x - radius, center_y - radius);
106
107         groundPath.moveTo(center_x - radius, center_y + radius);
108         groundPath.lineTo(center_x + radius, center_y + radius);
109         groundPath.lineTo(center_x + radius, y2);
110         groundPath.lineTo(center_x - radius, y1);
111         groundPath.lineTo(center_x - radius, center_y + radius);
112
113     }
114
115     painter.setBrush(QColor(135, 206, 235));
116     painter.drawPath(skyPath);
117     painter.setBrush(QColor(155, 118, 83));
118     painter.drawPath(groundPath);
119
120     // Disegna le linee orizzontali fisse ogni 10 gradi di pitch
121     painter.setPen(QPen(Qt::yellow, 2));
122     for (int i = -30; i <= 30; i += 10) {
123         int yLineOffset = static_cast<int>(center_y + tan(i * M_PI / 180.0) *
    radius);
124         if (yLineOffset >= center_y - radius && yLineOffset <= center_y +
    radius) {
125             painter.drawLine(QPoint(center_x - radius / 2 + (60-abs(i)),
    yLineOffset), QPoint(center_x + radius / 2 - (60-abs(i)), yLineOffset));
126             QString pitchLineText = QString("%1").arg(i);
127             painter.drawText(QPoint(center_x - radius / 2 + (30-abs(i)),
    yLineOffset + 5), pitchLineText);
128             painter.drawText(QPoint(center_x + radius / 2 - (50-abs(i)),
    yLineOffset + 5), pitchLineText);
129         }
130     }
131
132     // Disegna le tacche per l'angolo di roll
133     painter.setPen(QPen(Qt::yellow, 2));
134     for (int i = -30; i <= 30; i += 10) {
135         //double angle_rad = (i - roll) * M_PI / 180.0;
136         double angle_rad = (i) * M_PI / 180.0-M_PI/2;
137
138         int x1 = static_cast<int>(center_x - radius * cos(angle_rad));
139         int y1 = static_cast<int>(center_y + radius * sin(angle_rad));
140         int x2 = static_cast<int>(center_x - (radius - 10) * cos(angle_rad));
141         int y2 = static_cast<int>(center_y + (radius - 10) * sin(angle_rad));
142         painter.drawLine(x1, y1, x2, y2);
143
144         /*if (i % 30 == 0) { // Disegna l'etichetta ogni 30 gradi
145             QString rollLineText = QString("%1").arg(i);
146             painter.drawText(QPoint(x2 - 5, y2 + 15), rollLineText);
147         }*/
148     }
149
150
151
152     double angle_rad = (roll-3) * M_PI / 180.0 - M_PI/2;
153     QPoint point1(center_x + (radius-17) * cos(angle_rad), center_y + (radius
    -17) * sin(angle_rad));
154
155     angle_rad = (roll) * M_PI / 180.0 - M_PI/2;
156     QPoint point2(center_x + (radius-12) * cos(angle_rad), center_y + (radius
    -12) * sin(angle_rad));
157
158     angle_rad = (roll+3) * M_PI / 180.0 - M_PI/2;
159     QPoint point3(center_x + (radius-17) * cos(angle_rad), center_y + (radius
    -17) * sin(angle_rad));
160
```

```
161     // Crea un array di punti
162     QPoint points[3] = { point1, point3, point2 };
163 
164     painter.setBrush(Qt::yellow); // Colore di riempimento
165 
166     // Imposta il colore e lo stile del bordo del triangolo
167     //painter.setPen(QPen(Qt::black, 2));
168 
169     painter.drawConvexPolygon(points, 3);
170 
171     // Imposta il colore del testo
172     painter.setPen(Qt::blue);
173 
174     // Mostra gli angoli in sovraimpressione
175     QString pitchText = QString("Pitch: %1°").arg(pitch_d * 180 / M_PI, 0, 'f'
        , 1); // Converti in gradi e formatta con una cifra decimale
176     QString rollText = QString("Roll: %1°").arg(roll_d * 180 / M_PI, 0, 'f',
        1);  // Converti in gradi e formatta con una cifra decimale
177     QString yawText = QString("Yaw: %1°").arg(yaw_d * 180 / M_PI, 0, 'f', 1);
           // Converti in gradi e formatta con una cifra decimale
178 
179     QFont font = painter.font();
180     font.setPointSize(16); // Imposta la dimensione del font a 16 punti
181     painter.setFont(font);
182 
183     painter.drawText(40, 80, pitchText);  // Posiziona il testo del pitch in
        alto a sinistra
184     painter.drawText(40, 100, rollText);   // Posiziona il testo del roll
        sotto il pitch
185     painter.drawText(40, 120, yawText);    // Posiziona il testo del yaw sotto
         il roll
186 }
187 
188 void AttitudeWidget::updateAttitude()
189 {
190     update();
191 }
192 
193 void AttitudeWidget::set_roll(short roll){
194     this->roll = roll;
195 }
196 void AttitudeWidget::set_pitch(short pitch){
197     this->pitch = pitch;
198 }
199 void AttitudeWidget::set_yaw(short yaw){
200     this->yaw = yaw;
201 }
202 short AttitudeWidget::get_roll() const{
203     return this->roll;
204 }
205 short AttitudeWidget::get_pitch() const{
206     return this->pitch;
207 }
208 short AttitudeWidget::get_yaw() const{
209     return this->yaw;
210 }
```

A.3.5.7 messagereceiver.h

```
1  #ifndef MESSAGERECEIVER_H
2  #define MESSAGERECEIVER_H
3 
4 
5  #include "FACE_LAYERS/TSS/TS/FaceMessage_TS.hpp"
6  #include "ace/Log_Msg.h"
7 
8  #ifdef ACE_AS_STATIC_LIBS
9  # include "dds/DCPS/RTPS/RtpsDiscovery.h"
10 # include "dds/DCPS/transport/rtps_udp/RtpsUdp.h"
11 #endif
12 
```

```
#include <QtCore/QObject>
#include <QtCore/QThread>
#include <QtCore/QDebug>

#include "flag.h"

// #include "FACE/TS.hpp" // Assicurati di includere il percorso corretto

class MessageReceiver : public QObject
{
    Q_OBJECT

public:
    explicit MessageReceiver(QObject *parent = nullptr);
    ~MessageReceiver();

public slots:
    void receiveMessages();

signals:
#ifdef USE_TSS
    void messageReceived(Messenger::Position_DF msg);
#else
    void messageReceived(FACE::DF2ACP_message_t msg);
#endif
    void messageError(FACE::RETURN_CODE_TYPE status);

private:

    FACE::RETURN_CODE_TYPE status;
#ifdef USE_TSS

    // from DF via TSS
    Messenger::Position_DF msg_TS_df;
    FACE::CONNECTION_ID_TYPE connId_df;
    FACE::TRANSACTION_ID_TYPE txn_df;
    FACE::CONNECTION_DIRECTION_TYPE dir_df;
    FACE::MESSAGE_SIZE_TYPE max_msg_size_df;
    const FACE::TIMEOUT_TYPE timeout = 1.5e8;

#else
    QTimer *timer;
    int fd;
    short offset;
#endif
};

#endif // MESSAGERECEIVER_H
```

A.3.5.8 messagereceiver.cpp

```
#include "messagereceiver.h"

MessageReceiver::MessageReceiver(QObject *parent)
    : QObject(parent)
{

 #ifdef USE_TSS
    FACE::TS::Initialize("FACE_LAYERS/TSS/TS/config_static_PSGS.ini", status);

    FACE::TS::Create_Connection(
        "sub_DF", FACE::PUB_SUB, connId_df, dir_df,
        max_msg_size_df, FACE::INF_TIME_VALUE, status);

    qRegisterMetaType<Messenger::Position_DF>("Messenger::Position_DF");


#else
    timer = new QTimer(this);
    connect(timer, &QTimer::timeout, this, &MessageReceiver::receiveMessages);
    offset = 0;
```

```
    fd = open("FACE_LAYERS/PCS/log.txt", O_RDONLY, 0777);
    if(fd<0){
        printf("File NOT found!");
    }
    qRegisterMetaType<FACE::DF2ACP_message_t>("FACE::DF2ACP_message_t");
    timer->start(15);
#endif

}


void MessageReceiver::receiveMessages() {
#ifdef USE_TSS
    while (true) {
        FACE::TS::Receive_Message(connId_df, 1.5e8, txn_df, msg_TS_df,
    max_msg_size_df, status);
        printf("Msg received\n");
        if (status == FACE::NO_ERROR) {
            emit messageReceived(msg_TS_df);
            printf("Messaggio di ACPA ricevuta mediante TSS\n");
            fflush(stdout);
        }
        else if(status==FACE::INVALID_CONFIG){

            FACE::TS::Initialize("FACE_LAYERS/TSS/TS/config_static_PSGS.ini",
    status);
            FACE::TS::Create_Connection(
                "sub_DF", FACE::PUB_SUB, connId_df, dir_df,
                max_msg_size_df, FACE::INF_TIME_VALUE, status);

            // maybe next line
            // qRegisterMetaType<Messenger::Position_DF>("Messenger::
    Position_DF");

        }else {
            emit messageError(status);
        }
        usleep(1e4);
    }
#else

    off_t ris = lseek(fd, sizeof(FACE::DF2ACP_message_t)*offset, SEEK_SET);
    read(fd, (void*) &msg, sizeof(FACE::DF2ACP_message_t));
    //printf("Msg read...\n");
    fflush(stdout);
    offset++;
    //status = FACE::NO_ERROR;
    emit messageReceived(msg);

    printf("%f %f\n",  msg.pos.lat,  msg.pos.lon);
#endif
}

MessageReceiver::~MessageReceiver(){
#ifdef USE_TSS
    FACE::TS::Destroy_Connection(connId_df, status);
#else
    ::close(fd);
#endif
}
```

A.3.5.9 mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "unistd.h"
#include "fcntl.h"
#include "right_screen.h"
#include "left_screen.h"
```

```
#include "messagereceiver.h"

#include <QTimer>

#include "FACE_LAYERS/TSS/TS/FaceMessage_TS.hpp"
#include "ace/Log_Msg.h"
#include "ace/OS_NS_unistd.h"

#ifdef ACE_AS_STATIC_LIBS
# include "dds/DCPS/RTPS/RtpsDiscovery.h"
# include "dds/DCPS/transport/rtps_udp/RtpsUdp.h"
#endif

/*typedef struct HM_APP_s{
    short df;
    short egi;
    short gps;
}HM_APP_t;*/

QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

/*
protected:
    void paintEvent(QPaintEvent *event) override;
*/

private slots:
#ifdef USE_TSS
    void updateInfo(Messenger::Position_DF msg_TS_df);
#else
    void updateInfo(FACE::DF2ACP_message_t msg);
#endif
    //void showError(FACE::RETURN_CODE_TYPE status);

    void checkHM();

private:
    Ui::MainWindow *ui;
    QSplitter *splitter;
    QGridLayout *layout;
    QWidget *leftPanel;
    QWidget *rightPanel;
    PositionWidget *posWidget;
    AttitudeWidget *attWidget;

    QTimer *HM_check_timer;

#ifdef USE_TSS
    MessageReceiver* receiver;
#else
    FACE::DF2ACP_message_t msg;
#endif
    FACE::HM_APP_t hm_msg;

    FACE::RETURN_CODE_TYPE status;

};
#endif // MAINWINDOW_H
```

A.3.5.10 mainwindow.cpp

```
#include "mainwindow.h"
#include "./ui_mainwindow.h"
#include "stdio.h"
#include "stdlib.h"
#include "fcntl.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);


    QThread* thread = new QThread();
    receiver = new MessageReceiver();
    receiver->moveToThread(thread);

    connect(thread, &QThread::started, receiver, &MessageReceiver::
    receiveMessages);
    connect(receiver, &MessageReceiver::messageReceived, this, &MainWindow::
    updateInfo);
    //connect(receiver, &MessageReceiver::messageError, this, &MainWindow::
    showError);

    thread->start();

    //timerLeft = new QTimer(this);
    //timerRight = new QTimer(this);

    HM_check_timer = new QTimer(this);

    //connect(timerLeft, &QTimer::timeout, this, &MainWindow::updateInfo);
    //timerLeft->start(10);

    connect(HM_check_timer, &QTimer::timeout, this, &MainWindow::checkHM);
    HM_check_timer->start(5000);

    splitter = new QSplitter(this);

    QWidget *w = new QWidget(this);

    //layout = new QGridLayout(w);

    leftPanel = new QWidget(this);
    rightPanel = new QWidget(this);
    posWidget = new PositionWidget(this);
    attWidget = new AttitudeWidget(this);

    QFrame *f1 = new QFrame(this);

    f1->setFrameStyle(QFrame::Box | QFrame::Raised);
    f1->setLineWidth(2);
    f1->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);

    f1->setLayout(new QVBoxLayout(this));

    QFont font;
    QLabel *attitude_label = new QLabel("ATTITUDE DATA");
    font.setBold(true); attitude_label->setFont(font);


    attitude_label->setFixedHeight(20);
    f1->layout()->addWidget(attitude_label);
    f1->layout()->addWidget(attWidget);

    splitter->addWidget(posWidget);
    splitter->addWidget(f1);
    splitter->setStretchFactor(0, 1);
    splitter->setStretchFactor(1, 10);

    //layout->addWidget(posWidget, 0, 0, 1, 1);
    //layout->addWidget(attWidget, 0, 2, 1, 1);

```

```
71      setCentralWidget(splitter);
72
73  }
74
75  #ifdef USE_TSS
76  void MainWindow::updateInfo(Messenger::Position_DF msg_TS_df){
77      switch(msg_TS_df.fuse_src){
78          case FACE::FUSION_TYPE::BOTH:{
79              //printf("BOTH\n");
80              fflush(stdout);
81              double lat = msg_TS_df.pos.lat;
82              double lon = msg_TS_df.pos.lon;
83              int alt = msg_TS_df.pos.alt;
84
85              float heading = msg_TS_df.att.heading;
86              short pitch = msg_TS_df.att.pitch;
87              short roll = msg_TS_df.att.roll;
88              short yaw = msg_TS_df.att.yaw;
89
90              double ac_speed = msg_TS_df.speed;
91
92              speed_t speed;
93              acc_t acc;
94
95              speed.X = msg_TS_df.speed_NED.X;
96              speed.Y = msg_TS_df.speed_NED.Y;
97              speed.Z = msg_TS_df.speed_NED.Z;
98
99              acc.X = msg_TS_df.acc_NED.X;
100             acc.Y = msg_TS_df.acc_NED.Y;
101             acc.Z = msg_TS_df.acc_NED.Z;
102
103             this->posWidget->set_heading(heading);
104             this->posWidget->set_alt(alt);
105             this->posWidget->set_lat(lat);
106             this->posWidget->set_lon(lon);
107
108             //EGI
109             this->posWidget->set_lat(msg_TS_df.egi.pos.lat, 0);
110             this->posWidget->set_lon(msg_TS_df.egi.pos.lon, 0);
111             //GPS
112             this->posWidget->set_lat(msg_TS_df.gps.pos.lat, 1);
113             this->posWidget->set_lon(msg_TS_df.gps.pos.lon, 1);
114
115             //printf("%f %f %f %f\n", msg.egi.pos.lat, msg.gps.pos.lat, msg.
    egi.pos.lon, msg.gps.pos.lon);
116
117
118             this->posWidget->set_ac_speed(ac_speed);
119             this->posWidget->set_speed(speed);
120             this->posWidget->set_acc(acc);
121
122             this->attWidget->set_pitch(pitch);
123             this->attWidget->set_roll(roll);
124             this->attWidget->set_yaw(yaw);
125
126             this->posWidget->set_yaw(yaw);
127
128             this->posWidget->setEgiRead(true);
129             this->posWidget->setGpsRead(true);
130
131             this->posWidget->updatePosition();
132             this->attWidget->updateAttitude();
133
134             break;
135         }
136         case FACE::FUSION_TYPE::EGI:{
137             printf("EGI\n");
138             fflush(stdout);
139             double lat = msg_TS_df.egi.pos.lat;
140             double lon = msg_TS_df.egi.pos.lon;
141             int alt = msg_TS_df.egi.pos.alt;
142
143             float heading = msg_TS_df.egi.att.heading;
144             short pitch = msg_TS_df.egi.att.pitch;
```

```
145             short roll = msg_TS_df.egi.att.roll;
146             short yaw = msg_TS_df.egi.att.yaw;
147 
148             double ac_speed = msg_TS_df.egi.speed;
149 
150             speed_t speed;
151             acc_t acc;
152 
153             speed.X = msg_TS_df.egi.speed_NED.X;
154             speed.Y = msg_TS_df.egi.speed_NED.Y;
155             speed.Z = msg_TS_df.egi.speed_NED.Z;
156 
157             acc.X = msg_TS_df.egi.acc_NED.X;
158             acc.Y = msg_TS_df.egi.acc_NED.Y;
159             acc.Z = msg_TS_df.egi.acc_NED.Z;
160 
161             this->posWidget->set_heading(heading);
162             this->posWidget->set_alt(alt);
163             this->posWidget->set_lat(lat);
164             this->posWidget->set_lon(lon);
165             this->posWidget->set_lat(lat, 0);
166             this->posWidget->set_lon(lon, 0);
167 
168             this->posWidget->set_ac_speed(ac_speed);
169             this->posWidget->set_speed(speed);
170             this->posWidget->set_acc(acc);
171 
172             this->attWidget->set_pitch(pitch);
173             this->attWidget->set_roll(roll);
174             this->attWidget->set_yaw(yaw);
175 
176             this->posWidget->setEgiRead(true);
177             this->posWidget->setGpsRead(false);
178 
179             this->posWidget->updatePosition();
180             this->attWidget->updateAttitude();
181 
182             break;
183         }
184         case FACE::FUSION_TYPE::GPS:{
185             printf("GPS\n");
186             fflush(stdout);
187             double lat = msg_TS_df.gps.pos.lat;
188             double lon = msg_TS_df.gps.pos.lon;
189             int alt = msg_TS_df.gps.pos.alt;
190 
191             float heading = msg_TS_df.gps.att.heading;
192 
193             double ac_speed = msg_TS_df.gps.speed;
194 
195             this->posWidget->set_heading(heading);
196             this->posWidget->set_alt(alt);
197             this->posWidget->set_lat(lat);
198             this->posWidget->set_lon(lon);
199             this->posWidget->set_lat(lat, 1);
200             this->posWidget->set_lon(lon, 1);
201 
202             this->posWidget->set_ac_speed(ac_speed);
203 
204             this->posWidget->setEgiRead(false);
205             this->posWidget->setGpsRead(true);
206 
207             this->posWidget->updatePosition();
208             this->attWidget->updateAttitude();
209 
210             break;
211         }
212         case FACE::FUSION_TYPE::NONE:{
213             this->posWidget->setEgiRead(false);
214             this->posWidget->setGpsRead(false);
215 
216             this->posWidget->updatePosition();
217             this->attWidget->updateAttitude();
218             break;
219         }
```

```
220     }
221 #else
222 void MainWindow::updateInfo(FACE::DF2ACP_message_t msg){
223     switch(msg.fusion){
224         case FACE::FUSION_TYPE::BOTH:{
225             //printf("BOTH\n");
226             fflush(stdout);
227             double lat = msg.pos.lat;
228             double lon = msg.pos.lon;
229             int alt = msg.pos.alt;
230 
231             float heading = msg.att.hd;
232             short pitch = msg.att.pitch;
233             short roll = msg.att.roll;
234             short yaw = msg.att.yaw;
235 
236             double ac_speed = msg.ac_speed;
237 
238             speed_t speed;
239             acc_t acc;
240 
241             speed.X = msg.speed.X;
242             speed.Y = msg.speed.Y;
243             speed.Z = msg.speed.Z;
244 
245             acc.X = msg.acc.X;
246             acc.Y = msg.acc.Y;
247             acc.Z = msg.acc.Z;
248 
249             this->posWidget->set_heading(heading);
250             this->posWidget->set_alt(alt);
251             this->posWidget->set_lat(lat);
252             this->posWidget->set_lon(lon);
253 
254             //EGI
255             this->posWidget->set_lat(msg.egi.pos.lat, 0);
256             this->posWidget->set_lon(msg.egi.pos.lon, 0);
257             //GPS
258             this->posWidget->set_lat(msg.gps.pos.lat, 1);
259             this->posWidget->set_lon(msg.gps.pos.lon, 1);
260 
261             //printf("%f %f %f %f\n", msg.egi.pos.lat, msg.gps.pos.lat, msg.
    egi.pos.lon, msg.gps.pos.lon);
262 
263 
264             this->posWidget->set_ac_speed(ac_speed);
265             this->posWidget->set_speed(speed);
266             this->posWidget->set_acc(acc);
267 
268             this->attWidget->set_pitch(pitch);
269             this->attWidget->set_roll(roll);
270             this->attWidget->set_yaw(yaw);
271 
272             this->posWidget->set_yaw(yaw);
273 
274             this->posWidget->setEgiRead(true);
275             this->posWidget->setGpsRead(true);
276 
277             this->posWidget->updatePosition();
278             this->attWidget->updateAttitude();
279 
280             break;
281         }
282         case FACE::FUSION_TYPE::EGI:{
283             printf("EGI\n");
284             fflush(stdout);
285             double lat = msg.egi.pos.lat;
286             double lon = msg.egi.pos.lon;
287             int alt = msg.egi.pos.alt;
288 
289             float heading = msg.egi.att.hd;
290             short pitch = msg.egi.att.pitch;
291             short roll = msg.egi.att.roll;
292             short yaw = msg.egi.att.yaw;
293 
```

```
294             double ac_speed = msg.egi.ac_speed;
295 
296             speed_t speed;
297             acc_t acc;
298 
299             speed.X = msg.egi.speed.X;
300             speed.Y = msg.egi.speed.Y;
301             speed.Z = msg.egi.speed.Z;
302 
303             acc.X = msg.egi.acc.X;
304             acc.Y = msg.egi.acc.Y;
305             acc.Z = msg.egi.acc.Z;
306 
307             this->posWidget->set_heading(heading);
308             this->posWidget->set_alt(alt);
309             this->posWidget->set_lat(lat);
310             this->posWidget->set_lon(lon);
311             this->posWidget->set_lat(lat, 0);
312             this->posWidget->set_lon(lon, 0);
313 
314             this->posWidget->set_ac_speed(ac_speed);
315             this->posWidget->set_speed(speed);
316             this->posWidget->set_acc(acc);
317 
318             this->attWidget->set_pitch(pitch);
319             this->attWidget->set_roll(roll);
320             this->attWidget->set_yaw(yaw);
321 
322             this->posWidget->setEgiRead(true);
323             this->posWidget->setGpsRead(false);
324 
325             this->posWidget->updatePosition();
326             this->attWidget->updateAttitude();
327 
328             break;
329         }
330         case FACE::FUSION_TYPE::GPS:{
331             printf("GPS\n");
332             fflush(stdout);
333             double lat = msg.gps.pos.lat;
334             double lon = msg.gps.pos.lon;
335             int alt = msg.gps.pos.alt;
336 
337             float heading = msg.gps.att.hd;
338 
339             double ac_speed = msg.gps.speed;
340 
341             this->posWidget->set_heading(heading);
342             this->posWidget->set_alt(alt);
343             this->posWidget->set_lat(lat);
344             this->posWidget->set_lon(lon);
345             this->posWidget->set_lat(lat, 1);
346             this->posWidget->set_lon(lon, 1);
347 
348             this->posWidget->set_ac_speed(ac_speed);
349 
350             this->posWidget->setEgiRead(false);
351             this->posWidget->setGpsRead(true);
352 
353             this->posWidget->updatePosition();
354             this->attWidget->updateAttitude();
355 
356             break;
357         }
358         case FACE::FUSION_TYPE::NONE:{
359             this->posWidget->setEgiRead(false);
360             this->posWidget->setGpsRead(false);
361 
362             this->posWidget->updatePosition();
363             this->attWidget->updateAttitude();
364             break;
365         }
366     }
367 
368 #endif
```

```
}

void MainWindow::checkHM(){
#ifdef USE_TSS
    //ser_hm.Read(handle_hm, FACE::NO_WAIT, &rd, sizeof(FACE::HM_APP_t),
    status);
    char *buf;
    this->posWidget->read_serial(buf, status);
    if(status==FACE::NO_ERROR){
        memcpy((void*) &hm_msg, (void*) buf, sizeof(FACE::HM_APP_t));
        this->posWidget->set_DF_APP(!(bool)hm_msg.df);
        this->posWidget->set_EGI_APP(!(bool)hm_msg.egi);
        this->posWidget->set_GPS_APP(!(bool)hm_msg.gps);
    }
#endif
}

MainWindow::~MainWindow(){
#ifdef USE_TSS
    delete this->receiver;
#else
    ::close(fd);
#endif
    delete ui;
}
```

A.3.5.11 main.cpp

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.setWindowTitle("COCKPIT SIMULATOR");
    w.setFixedSize(1100, 800);
    w.show();
    return a.exec();
}
```

A.3.6 Device Protocol Mediation

A.3.6.1 backup.h

```
#ifndef BACKUP_H
#define BACKUP_H

#include "time.h"
#include "../sim_utils.h"
#include "FACE_LAYERS/IOSS/Common.h"

#define MAX_RECORDS_NUMBER 1024

typedef struct position_s{
    double lat;
    double lon;
    int alt;
}position_t;

typedef struct attitude_s{
    short pitch;
    short roll;
    float hd;
```

```
    short yaw;
}attitude_t;

typedef struct backup_s{
    time_t timestamp;
    position_t pos[MAX_RECORDS_NUMBER];
    attitude_t att[MAX_RECORDS_NUMBER];
    unsigned short idx;
}backup_t;

// struct similar to TSS message
typedef struct messaggio_EGI_s{
    FACE::TIMEOUT_TYPE tmp;
    position_t pos;
    attitude_t att;
    speed_t speed;
    acc_t acc;
    double ac_speed;
}messaggio_EGI_t;

// struct similar to TSS message
typedef struct messaggio_GPS_s{
    FACE::TIMEOUT_TYPE tmp;
    position_t pos;
    attitude_t att;
    double speed;
}messaggio_GPS_t;


#endif // BACKUP_H
```

A.3.6.2 DMP.h

```
#ifndef FTP_BACKUP_H
#define FTP_BACKUP_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netdb.h>

#include "AC_SIM/sim_constant.h"

#define BUFFER_SIZE 1024

int connect_to_server(const char *hostname, int port);
void send_command(int sockfd, const char *command);
void read_response(int sockfd, char *buffer, size_t size);
int open_data_connection(int sockfd);
void list_files(int control_sockfd);
void retrieve_file(int control_sockfd, const char *filename);
void store_file(int control_sockfd, const char *filename);


#endif // FTP_BACKUP_H
```

A.3.6.3 DPM.cpp

```
#include "DPM.h"

#define BUFFER_SIZE 1024

int connect_to_server(const char *hostname, int port) {
    struct sockaddr_in server_addr;
```

```
7       struct hostent *server;
8
9       int sockfd = socket(AF_INET, SOCK_STREAM, 0);
10      if (sockfd < 0) {
11          perror("ERROR opening socket");
12          exit(EXIT_FAILURE);
13      }
14
15      server = gethostbyname(hostname);
16      if (server == NULL) {
17          fprintf(stderr, "ERROR, no such host\n");
18          exit(EXIT_FAILURE);
19      }
20
21      memset((char *) &server_addr, 0, sizeof(server_addr));
22      server_addr.sin_family = AF_INET;
23      memcpy((char *) server->h_addr, (char *) &server_addr.sin_addr.s_addr,
        server->h_length);
24      server_addr.sin_port = htons(port);
25
26      if (connect(sockfd, (struct sockaddr *) &server_addr, sizeof(server_addr))
         < 0) {
27          perror("ERROR connecting");
28          close(sockfd);
29          exit(EXIT_FAILURE);
30      }
31
32      return sockfd;
33  }
34
35  void send_command(int sockfd, const char *command) {
36      char buffer[BUFFER_SIZE];
37      snprintf(buffer, sizeof(buffer), "%s\r\n", command);
38      write(sockfd, buffer, strlen(buffer));
39  }
40
41  void read_response(int sockfd, char *buffer, size_t size) {
42      memset(buffer, 0, size);
43      read(sockfd, buffer, size - 1);
44  }
45
46  int open_data_connection(int sockfd) {
47      char buffer[BUFFER_SIZE];
48      char *token;
49      int data_port;
50      int data_sockfd;
51      struct sockaddr_in data_addr;
52
53      send_command(sockfd, "PASV");
54      read_response(sockfd, buffer, sizeof(buffer));
55      printf("PASV response: %s", buffer);
56
57
58      token = strtok(buffer, "(,");
59      for (int i = 0; i < 4; i++) {
60          token = strtok(NULL, "(,");
61      }
62
63      int ip_part0 = atoi(token);
64      token = strtok(NULL, "(,");
65      int port_part1 = atoi(token);
66      token = strtok(NULL, "(,");
67      int port_part2 = atoi(token);
68      token = strtok(NULL, "(,");
69
70      data_port = (port_part1 << 8) | port_part2;
71
72      data_sockfd = socket(AF_INET, SOCK_STREAM, 0);
73      if (data_sockfd < 0) {
74          perror("ERROR opening data socket");
75          exit(EXIT_FAILURE);
76      }
77
78      memset((char *) &data_addr, 0, sizeof(data_addr));
79      data_addr.sin_family = AF_INET;
```

```
80      data_addr.sin_addr.s_addr = inet_addr("127.0.0.1"); // Use the IP address
        obtained from PASV response
81
82      printf("Response PORT: %d", data_port);
83
84      data_addr.sin_port = htons(data_port);
85
86      if (connect(data_sockfd, (struct sockaddr *) &data_addr, sizeof(data_addr)
        ) < 0) {
87          perror("ERROR connecting to data socket");
88          close(data_sockfd);
89          exit(EXIT_FAILURE);
90      }
91
92      return data_sockfd;
93  }
94
95  void list_files(int control_sockfd) {
96      char buffer[BUFFER_SIZE];
97      int data_sockfd = open_data_connection(control_sockfd);
98
99      send_command(control_sockfd, "LIST");
100     read_response(control_sockfd, buffer, sizeof(buffer));
101     printf("%s", buffer);
102     memset(buffer, 0, sizeof(buffer));
103     while (read(data_sockfd, buffer, sizeof(buffer) - 1) > 0) {
104         printf("%s", buffer);
105         memset(buffer, 0, sizeof(buffer));
106     }
107
108     close(data_sockfd);
109 }
110
111 void retrieve_file(int control_sockfd, const char *filename) {
112     char buffer[BUFFER_SIZE];
113     int data_sockfd = open_data_connection(control_sockfd);
114
115     snprintf(buffer, sizeof(buffer), "RETR %s", filename);
116     send_command(control_sockfd, buffer);
117     read_response(control_sockfd, buffer, sizeof(buffer));
118     printf("%s", buffer);
119
120     FILE *file = fopen(filename, "wb");
121     if (file == NULL) {
122         perror("ERROR opening file");
123         close(data_sockfd);
124         return;
125     }
126
127     memset(buffer, 0, sizeof(buffer));
128     int bytes_read;
129     while ((bytes_read = read(data_sockfd, buffer, sizeof(buffer) - 1)) > 0) {
130         fwrite(buffer, 1, bytes_read, file);
131         memset(buffer, 0, sizeof(buffer));
132     }
133
134     fclose(file);
135     close(data_sockfd);
136 }
137
138 void store_file(int control_sockfd, const char *filename) {
139     char buffer[BUFFER_SIZE];
140     int data_sockfd = open_data_connection(control_sockfd);
141
142     snprintf(buffer, sizeof(buffer), "STOR %s", filename);
143     send_command(control_sockfd, buffer);
144     read_response(control_sockfd, buffer, sizeof(buffer));
145     printf("%s", buffer);
146
147     FILE *file = fopen(filename, "rb");
148     if (file == NULL) {
149         perror("ERROR opening file");
150         close(data_sockfd);
151         return;
152     }
```

```
153
154     memset(buffer, 0, sizeof(buffer));
155     int bytes_read;
156     while ((bytes_read = fread(buffer, 1, sizeof(buffer) - 1, file)) > 0) {
157         write(data_sockfd, buffer, bytes_read);
158         memset(buffer, 0, sizeof(buffer));
159     }
160
161     fclose(file);
162     close(data_sockfd);
163 }
```

A.3.6.4 main.cpp

```
1  #include <iostream>
2
3  #include "FACE_LAYERS/IOSS/Serial/FACE_Serial.h"
4  #include "FACE_LAYERS/IOSS/Common.h"
5  #include "DPM.h"
6  #include "../sim_utils.h"
7  #include "../EGI/EGI.h"
8  #include "../GPS/GPS.h"
9  #include "backup.h"
10
11 void retrieve_file_from_server(){
12     const char *hostname = "localhost";
13     const char *username = "edoardo";
14     const char *password = "pwd_to_access";
15
16     int control_sockfd = connect_to_server(hostname, 21);
17     char buffer[BUFFER_SIZE];
18     read_response(control_sockfd, buffer, sizeof(buffer));
19
20     snprintf(buffer, (6+strlen(username)), "USER %s", username);
21     send_command(control_sockfd, buffer);
22     read_response(control_sockfd, buffer, sizeof(buffer));
23
24     snprintf(buffer, (6+strlen(password)), "PASS %s", password);
25     send_command(control_sockfd, buffer);
26     read_response(control_sockfd, buffer, sizeof(buffer));
27
28     retrieve_file(control_sockfd, "backup.txt");
29     close(control_sockfd);
30 }
31
32 void *DPM_EGI(void *p){
33     FACE_Serial serial;
34     FACE::RETURN_CODE_TYPE return_code;
35     FACE::CONFIGURATION_RESOURCE config = "FACE_LAYERS/PSSS/DPM/EGI";
36
37     FACE::IOSS::Serial::CONNECTION_NAME_TYPE name;
38     FACE::IOSS::CONNECTION_HANDLE_TYPE handle;
39     FACE::IOSS::Serial::READ_PAYLOAD_TYPE rd;
40     FACE::IOSS::Serial::WRITE_PAYLOAD_TYPE wr;
41
42 #ifdef COMPRESS
43     char* cmp_packet = (char*) malloc(sizeof(char)*EGI_PACKET_SIZE);
44 #endif
45
46     serial.Initialize(config, return_code);
47     if(return_code!=FACE::NO_ERROR){
48         perror("Error!!");
49     }
50
51     serial.Open_Connection(name, FACE::IOSS::Serial::WAIT_FOREVER, handle,
       return_code);
52     if(return_code!=FACE::NO_ERROR){
53         perror("Error!!");
54     }
55
56     while(1){
```

```
57
58          serial.Read(handle, FACE::IOSS::Serial::WAIT_FOREVER, &rd, 0,
    return_code);  //blocking
59
60          retrieve_file_from_server();
61
62          int fd = open("backup.txt", O_RDONLY);
63
64          backup_t bak;
65          read(fd, (void*) &bak, sizeof(bak));
66
67          uint16_t i=0;
68          while(i<bak.idx){
69  #ifdef COMPRESS
70              std::string c1_bin = numericToBinary<float>((float)((delta_lon+
    lon0)/MAX_LON), 24);
71              std::string c2_bin = numericToBinary<float>((float)((delta_lat+
    lat0)/MAX_LAT), 24);
72              std::string alt_bin = numericToBinary<float>(((float)(delta_alt+
    alt0)/MAX_ALTITUDE), 16);
73              std::string s_bin = numericToBinary<float>(speed/MAX_SPEED, 16);
74              std::string hd_bin = numericToBinary<float>(heading/(float)(M_PI),
     20);
75              std::string p_bin = numericToBinary<float>((float)pitch/(float)
    MAX_PITCH, 16);
76              std::string r_bin = numericToBinary<float>((float)roll/(float)
    MAX_ROLL, 16);
77              std::string y_bin = numericToBinary<float>((float)yaw/(float)
    MAX_YAW, 16);
78              std::string s_x_bin = numericToBinary<float>(speed_NED.X/MAX_SPEED
    , 20);
79              std::string s_y_bin = numericToBinary<float>(speed_NED.Y/MAX_SPEED
    , 20);
80              std::string s_z_bin = numericToBinary<float>(speed_NED.Z/MAX_SPEED
    , 20);
81              std::string a_x_bin = numericToBinary<float>(acc_NED.X/G, 16);
82              std::string a_y_bin = numericToBinary<float>(acc_NED.Y/G, 16);
83              std::string a_z_bin = numericToBinary<float>(acc_NED.Z/G, 16);
84
85              char *msg = (char*) calloc(EGI_PACKET_SIZE*8, sizeof(char));
86              strcat(msg, c1_bin.c_str());
87              strcat(msg, c2_bin.c_str());
88              strcat(msg, alt_bin.c_str());
89              strcat(msg, s_bin.c_str());
90              strcat(msg, hd_bin.c_str());
91              strcat(msg, p_bin.c_str());
92              strcat(msg, r_bin.c_str());
93              strcat(msg, y_bin.c_str());
94              strcat(msg, s_x_bin.c_str());
95              strcat(msg, s_y_bin.c_str());
96              strcat(msg, s_z_bin.c_str());
97              strcat(msg, a_x_bin.c_str());
98              strcat(msg, a_y_bin.c_str());
99              strcat(msg, a_z_bin.c_str());
100
101             compress(cmp_packet, msg, EGI_PACKET_SIZE);
102             memcpy(wr.payload, cmp_packet, EGI_PACKET_SIZE);
103
104             free(msg);
105 #else
106             std::string c1_bin = numericToBinary<double>(bak.pos[i].lat/
    MAX_LAT, 32);
107             std::string c2_bin = numericToBinary<double>(bak.pos[i].lon/
    MAX_LON, 32);
108             std::string h_bin = numericToBinary<float>((float)bak.pos[i].alt/
    MAX_ALTITUDE, 20);
109             memcpy(wr.payload, (c1_bin+c2_bin+h_bin).c_str(), EGI_PACKET_SIZE)
    ;
110 #endif
111             serial.Write(handle, FACE::IOSS::Serial::WAIT_FOREVER, &wr,
    EGI_PACKET_SIZE, return_code);
112             i++;
113         }
114
115         close(fd);
```

```
116     }
117
118     serial.Close_Connection(handle, return_code);
119     return NULL;
120
121 }
122 void *DPM_GPS(void *p){
123     FACE_Serial serial;
124     FACE::RETURN_CODE_TYPE return_code;
125     FACE::CONFIGURATION_RESOURCE config = "FACE_LAYERS/PSSS/DPM/GPS";
126
127     FACE::IOSS::Serial::CONNECTION_NAME_TYPE name;
128     FACE::IOSS::CONNECTION_HANDLE_TYPE handle;
129     FACE::IOSS::Serial::READ_PAYLOAD_TYPE rd;
130     FACE::IOSS::Serial::WRITE_PAYLOAD_TYPE wr;
131
132 #ifdef COMPRESS
133     char* cmp_packet = (char*) malloc(sizeof(char)*GPS_PACKET_SIZE);
134 #endif
135
136     serial.Initialize(config, return_code);
137     if(return_code!=FACE::NO_ERROR){
138         perror("Error!!");
139     }
140
141     serial.Open_Connection(name, FACE::IOSS::Serial::WAIT_FOREVER, handle,
    return_code);
142     if(return_code!=FACE::NO_ERROR){
143         perror("Error!!");
144     }
145
146     while(1){
147
148         serial.Read(handle, FACE::IOSS::Serial::WAIT_FOREVER, &rd, 0,
    return_code);  //blocking
149
150         retrieve_file_from_server();
151
152         int fd = open("backup.txt", O_RDONLY);
153
154         backup_t bak;
155         read(fd, (void*) &bak, sizeof(bak));
156
157         uint16_t i=0;
158         while(i<bak.idx){
159 #ifdef COMPRESS
160
161             char *msg = (char*) calloc(GPS_PACKET_SIZE*8, sizeof(char));
162
163             std::string c1_bin = numericToBinary<double>((dmsToDec_r(lon)/
    MAX_LON), 32);
164             std::string c2_bin = numericToBinary<double>((dmsToDec_r(lat)/
    MAX_LAT), 32);
165             std::string alt_bin = numericToBinary<float>(((float)altitude/
    MAX_ALTITUDE), 20);
166             std::string s_bin = numericToBinary<float>((float)speed/MAX_SPEED,
     16);
167             std::string hd_bin = numericToBinary<float>(heading/(float)(M_PI),
     16);
168
169             strcat(msg, c1_bin.c_str());
170             strcat(msg, c2_bin.c_str());
171             strcat(msg, alt_bin.c_str());
172             strcat(msg, s_bin.c_str());
173             strcat(msg, hd_bin.c_str());
174
175             compress(cmp_packet, (c1_bin+c2_bin+h_bin).c_str(),
    GPS_PACKET_SIZE);
176             memcpy(wr.payload, cmp_packet, GPS_PACKET_SIZE);
177
178             free(msg);
179 #else
180             std::string c1_bin = numericToBinary<double>(bak.pos[i].lat/
    MAX_LAT, 32);
```

```
181             std::string c2_bin = numericToBinary<double>(bak.pos[i].lon/
    MAX_LON, 32);
182             std::string h_bin = numericToBinary<float>((float)bak.pos[i].alt/
    MAX_ALTITUDE, 20);
183             memcpy(wr.payload, (c1_bin+c2_bin+h_bin).c_str(), GPS_PACKET_SIZE)
    ;
184 #endif
185             serial.Write(handle, FACE::IOSS::Serial::WAIT_FOREVER, &wr,
    GPS_PACKET_SIZE, return_code);
186             i++;
187         }
188 
189         close(fd);
190     }
191 
192     serial.Close_Connection(handle, return_code);
193     return NULL;
194 }
195 
196 int main() {
197 
198     pthread_t tid1;
199     pthread_t tid2;
200 
201     pthread_create(&tid1, NULL, DPM_EGI, NULL);
202     pthread_create(&tid1, NULL, DPM_GPS, NULL);
203 
204     pthread_join(tid1, NULL);
205     pthread_join(tid2, NULL);
206 
207     return 0;
208 }
```

A.4 IOSS

A.4.1 Utility

A.4.1.1 Common.h

```
1  //
2  // Created by Edoardo Cavallotti on 30/05/24.
3  //
4 
5 
6  #ifndef __FACE_COMMON
7  #define __FACE_COMMON
8 
9 
10 namespace FACE {
11 
12 #ifndef RETURN_CODE_TYPE_DEFINED
13 #define RETURN_CODE_TYPE_DEFINED
14 //! This enumeration defines the possible set of status codes which may be
15 //! returned by a method defined in the FACE API.
16 //! The first set of codes (through TIMED_OUT) is constrained to match the
17 //! set defined in the ARINC 653 standard.
18 
19 typedef enum RETURN_CODE_TYPE {
20     NO_ERROR, //! request valid and operation performed
21     NO_ACTION, //! status of system unaffected by request
22     NOT_AVAILABLE, //! no message was available
23     INVALID_PARAM, //! invalid parameter specified in request
24     INVALID_CONFIG, //! parameter incompatible with configuration
25     INVALID_MODE, //! request incompatible with current mode
26     TIMED_OUT, //! the time expired before the request could be filled
27     ADDR_IN_USE, //! address currently in use
28     PERMISSION_DENIED, //! no permission to send or connecting to wrong
    partition
```

```
    MESSAGE_STALE, //! current time - timestamp exceeds configured limits
    IN_PROGRESS, //! asynchronous connection in progress
    CONNECTION_CLOSED, //! connection was closed
    DATA_BUFFER_TOO_SMALL, //! Data Buffer was too small for message
    DATA_OVERFLOW //! A loss of messages due to data buffer overflow
}RETURN_CODE_TYPE;
#endif // RETURN_CODE_TYPE_DEFINED
//! This type is used to represent 64-bit signed integer with a
//! 1 nanosecond resolution.
#ifndef SYS_TYPE_VALUE_DEFINED
#define SYS_TYPE_VALUE_DEFINED
typedef long SYSTEM_TIME_TYPE;
#endif

#ifndef INF_TIME_VALUE_DEFINED
#define INF_TIME_VALUE_DEFINED
//! This type is used to represent an infinitely long time value.
//! It is often used to specify that the caller is willing to wait
//! forever for an operation to complete and does not wish to timeout.
static const long INF_TIME_VALUE = -1;
#endif

//! This type is used to represent an unbounded string of characters
typedef char* UNBOUNDED_STRING_TYPE;
//! This type is used to represent a bounded string of characters.
typedef char STRING_TYPE[256];
//! This string is used to specify the location of the configuration
//! resource in both the IOSS and TSS initialize functions. This may be local,
//! a file name reference, or a reference to a configuration service.
typedef STRING_TYPE CONFIGURATION_RESOURCE;
//! This type is used to represent a system address.
//native SYSTEM_ADDRESS_TYPE;
//! This type has a one nanosecond resolution.
typedef SYSTEM_TIME_TYPE TIMEOUT_TYPE;
//! This type is used to represent Globally Unique Identifiers.
typedef long long GUID_TYPE;

// This type is used to represent the message range, number of messages,
// and message type in the I/O and Transport APIs.
typedef int MESSAGE_RANGE_TYPE;

// This type is used to represent a system address.
//
// Note: This is expected to be a pointer type such as (void *) in C.
typedef void* SYSTEM_ADDRESS_TYPE;
};
#endif // __FACE_COMMON
```

A.4.1.2 IO.h

```
//
// Created by Edoardo Cavallotti on 30/05/24.
//

#ifndef __FACE_IOSS_IOS
#define __FACE_IOSS_IOS

/*#ifdef __cplusplus
extern "C"
#endif*/


#include "Common.h"
#include "fcntl.h"
#include "unistd.h"
#include "string.h"
#include <vector>

// new defined
#define MAX_CONNECTIONS 32
#define PAYLOAD_DATA_MSG_SIZE 2048
```

```
#define DATA_CHUNK_SIZE 512

using namespace std;



namespace FACE {

    typedef unsigned long UnsignedLong;

    namespace IOSS {
/*// The status of the I/O device bus, separate from the status of a
// single I/O connection or of a device attached to that bus.
//
// NOTE: Bus-specific status constants are defined in their respective
// namespaces.*/
        typedef unsigned short BUS_STATUS_TYPE;
/*// I/O parameters are used to programmatically query or change
// properties of an I/O connection or its underlying bus after
// initialization. This may be a necessary part of PSSS UoC logic,
// such as auto-negotiating serial baud rate, and is fundamentally
// different from the FACE Configuration Interface that maps to an
// underlying and existent configuration resource.
//
// Each I/O Service defines the specific I/O parameters it supports
// based on the I/O connection analogy and the I/O bus architecture.
// The declarations here are the basis of those specific definitions.
// Used to declare constants to represent supported I/O parameters.
//
// NOTE: Bus-specific parameter IDs are defined in their namespaces.*/
        typedef unsigned short IO_PARAMETER_ID_TYPE;
// Used by I/O functions to specify the I/O connection addressed
// by the handle.
        typedef long long CONNECTION_HANDLE_TYPE;


// INVALID_CONNECTION_HANDLE and IGNORED_CONNECTION_HANDLE as sentinel
// values.
        const CONNECTION_HANDLE_TYPE INVALID_CONNECTION_HANDLE = -1;
        const CONNECTION_HANDLE_TYPE IGNORED_CONNECTION_HANDLE = 0;
/*//module IO_Service_Module<struct PAYLOAD_DATA_MSG_TYPE> {
// All declarations are in this template module so that the
// instantiated module for each I/O Service has a fully qualified
// type that is distinct. This improves type safety in the resulting
// language mappings.
// The unique name of the I/O connection. It is passed to Open(I/O)
// and typically is used to assign values specified in the
// configuration resource given to Initialize(I/O).*/

        namespace Serial {
            typedef STRING_TYPE CONNECTION_NAME_TYPE;

// Enumeration of the status condition of the I/O connection.
            typedef enum IO_CONNECTION_STATUS_TYPE {
                NOT_OPEN, // initial state prior to opening the connection
                CONNECTING,// transient state where attempt is being made to
    open the connection
                READY, // connection ready for service; it can now accept data
     r/w operations
                BUSY, // connection has congestion and cannot accept more data
                DEGRADED // connection not fully operational; there is some
    kind
            } IO_CONNECTION_STATUS_TYPE;
// This status represents the health of an I/O connection and
// availability of messages. It does not represent the status of the
// I/O device associated with the connection.

            typedef unsigned short BUS_STATUS_TYPE;

            typedef struct CONNECTION_STATUS_TYPE {
                SYSTEM_TIME_TYPE last_message_time;
                bool message_available;
                IO_CONNECTION_STATUS_TYPE connection_status;
            } CONNECTION_STATUS_TYPE;
// List of possible types for an I/O parameter value.
```

```
94              typedef enum IO_PARAMETER_VALUE_TYPES_TYPE {
95                  FACE_SHORT,
96                  FACE_LONG,
97                  FACE_LONGLONG,
98                  FACE_USHORT,
99                  FACE_ULONG,
100                 FACE_ULONGLONG,
101                 FACE_FLOAT,
102                 FACE_DOUBLE,
103                 FACE_LONGDOUBLE,
104                 FACE_CHAR,
105                 FACE_BOOLEAN,
106                 FACE_OCTET
107             } IO_PARAMETER_VALUE_TYPES_TYPE;
108 // The value for an I/O parameter.
109 //
110 // NOTE: Bus-specific parameter values are defined in their
111 // namespaces.
112             typedef struct IO_PARAMETER_VALUE_TYPE {
113                 IO_PARAMETER_VALUE_TYPES_TYPE discriminator;
114                 union IO_P_V_values {
115                     short short_value;
116                     long long_value;
117                     long long longlong_value;
118                     unsigned short ushort_value;
119                     unsigned long ulong_value;
120                     unsigned long long ulonglong_value;
121                     float float_value;
122                     double double_value;
123                     long double longdouble_value;
124                     char char_value;
125                     bool boolean_value;
126                     unsigned char octet_value;
127                 } values;
128             } IO_PARAMETER_VALUE_TYPE;
129
130 // Represent a single (ID,value) pair and a list of pairs.
131             typedef struct IO_PARAMETER {
132                 IO_PARAMETER_ID_TYPE id;
133                 IO_PARAMETER_VALUE_TYPE value;
134             } IO_PARAMETER;
135             typedef vector <IO_PARAMETER> IO_PARAMETER_LIST;
136
137 // Used to pass configuration properties across the I/O Service
138 // Interface.
139             typedef struct IO_PARAMETER_TRANSACTION_TYPE {
140                 GUID_TYPE guid; // used to differentiate transactions
141                 IO_PARAMETER_LIST items;
142             } IO_PARAMETER_TRANSACTION_TYPE;
143
144
145 /*// The interface is designed to operate with a local or remote
146 // implementation. Thus the timeout parameter on the interfaces allow
147 // the user call to block while the request may be remotely processed.
148 // A value of NO_WAIT indicates the request should be made but there
149 // should be no waiting for a response before returning to the caller.
150 // In such a case the action may not occur immediately. A value of
151 // WAIT_FOREVER indicates waiting indefinitely.*/
152
153             const TIMEOUT_TYPE NO_WAIT = 0;
154             const TIMEOUT_TYPE WAIT_FOREVER = -1;
155 // Notification events are generated when the PSSS UoC has registered
156 // a handler callback (see Register_Notification_Event_Handler(I/O)).
157 // Enumeration of the type of the notification event.
158             typedef enum NOTIFICATION_EVENT_TYPE {
159                 DATA_READ_EVENT, // Data is available via Read()
160                 CONNECTION_CONFIG_CHANGE_EVENT, // Use
    Get_Connection_Configuration()
161                 // for new parameters
162                 CONNECTION_STATUS_CHANGE_EVENT, // Use Get_Connection_Status()
163                 BUS_CONFIG_CHANGE_EVENT, // Use Get_Bus_Configuration() for
    new parameters
164                 BUS_STATUS_CHANGE_EVENT // Use Get_Bus_Status()
165             } NOTIFICATION_EVENT_TYPE;
166
```

```
// Defines the function prototype for a notification callback.
//
// NOTE: 'handle' is IGNORED_CONNECTION_HANDLE when
// 'notification_event' is a bus event.
            namespace IO_Callback {
                void Process_Notification_Event(
                        CONNECTION_HANDLE_TYPE handle,
                        NOTIFICATION_EVENT_TYPE notification_event);
            };
//interface IO_Service {
// The Initialize(I/O) function allows the PSSS UoC to provide a
// configuration resource to use when initializing an I/O Service.

            typedef STRING_TYPE CONFIGURATION_RESOURCE;

            void Initialize(/*in*/ CONFIGURATION_RESOURCE config_resource,
/*out*/ RETURN_CODE_TYPE &return_code);

// The Open_Connection(I/O) function is used by the PSSS UoC to
// create a connection to an I/O device. A unique handle for the
// connection is returned on success, or INVALID_CONNECTION_HANDLE.
            void Open_Connection(
/*in*/ CONNECTION_NAME_TYPE name,
/*in*/ TIMEOUT_TYPE timeout,
/*out*/ CONNECTION_HANDLE_TYPE &handle,
/*out*/ RETURN_CODE_TYPE &return_code);

// The Close_Connection(I/O) function is used by the PSSS UoC to
// close a connection and release the addressed handle.
            void Close_Connection(
/*in*/ CONNECTION_HANDLE_TYPE handle,
/*out*/ RETURN_CODE_TYPE &return_code);

// This structure defines the data payload format for reads.
// received_time is the time stamp most closely associated with the
// last byte going into the buffer.
// If received_time is set to IGNORE_RECEIVED_TIME, it should be
// ignored.*/
            const SYSTEM_TIME_TYPE IGNORE_RECEIVED_TIME = 0;

            typedef char *PAYLOAD_DATA_MSG_TYPE;

            typedef struct READ_PAYLOAD_TYPE {
                SYSTEM_TIME_TYPE received_time;
                PAYLOAD_DATA_MSG_TYPE payload;
            } READ_PAYLOAD_TYPE;

// The Read(I/O) function allows the PSSS UoC to synchronously receive
// (poll for) payload data. It is also called by the PSSS UoC to
// asynchronously receive payload data when a registered notification
// callback receives a DATA_READ_EVENT.

            void Read(/*in*/ CONNECTION_HANDLE_TYPE handle,
                    /*in*/ TIMEOUT_TYPE timeout,
                    /*inout*/ READ_PAYLOAD_TYPE *payload,
                    /*out*/ RETURN_CODE_TYPE &return_code);

// This structure defines the data payload format for writes.

            typedef struct WRITE_PAYLOAD_TYPE {
                PAYLOAD_DATA_MSG_TYPE payload;
            } WRITE_PAYLOAD_TYPE;
// The Write(I/O) operation call allows the PSSS UoC to
// synchronously send payload data

            void Write(/*in*/ CONNECTION_HANDLE_TYPE handle,
                    /*in*/ TIMEOUT_TYPE timeout,
                    /*in*/ WRITE_PAYLOAD_TYPE payload,
                    /*out*/RETURN_CODE_TYPE &return_code);

// The Configure_Connection_Parameters(I/O) function allows the
// PSSS UoC to assign I/O parameters for a connection.
            void Configure_Connection_Parameters(/*in*/CONNECTION_HANDLE_TYPE
    handle,
                    /*in*/TIMEOUT_TYPE timeout,
```

```
241                     /*in*/IO_PARAMETER_TRANSACTION_TYPE parameters,
242                     /*out*/ RETURN_CODE_TYPE &return_code);
243
244 // The Get_Connection_Configuration(I/O) function allows the PSSS
245 // UoC to query I/O parameters for a connection.
246             void Get_Connection_Configuration(/*in*/CONNECTION_HANDLE_TYPE
    handle,
247                     /*in*/TIMEOUT_TYPE timeout,
248                     /*inout*/IO_PARAMETER_TRANSACTION_TYPE *parameters,
249                     /*out*/RETURN_CODE_TYPE &return_code);
250
251 // The Configure_Bus_Parameters(I/O) function allows the PSSS UoC
252 // to assign I/O parameters for the I/O bus underlying an
253 // I/O connection.
254             void Configure_Bus_Parameters(
255 /*in*/CONNECTION_HANDLE_TYPE handle,
256 /*in*/TIMEOUT_TYPE timeout,
257 /*in*/IO_PARAMETER_TRANSACTION_TYPE parameters,
258 /*out*/RETURN_CODE_TYPE &return_code);
259
260 // The Get_Bus_Configuration(I/O) function allows the PSSS UoC to
261 // query I/O parameters for the I/O bus underlying an I/O connection.
262             void Get_Bus_Configuration(
263 /*in*/CONNECTION_HANDLE_TYPE handle,
264 /*in*/TIMEOUT_TYPE timeout,
265 /*inout*/IO_PARAMETER_TRANSACTION_TYPE *parameters,
266 /*out*/RETURN_CODE_TYPE &return_code);
267
268 // The Get_Connection_Status(I/O) function allows the PSSS UoC to
269 // query for connection status information.
270             void Get_Connection_Status(
271 /*in*/CONNECTION_HANDLE_TYPE handle,
272 /*out*/CONNECTION_STATUS_TYPE *status,
273 /*out*/RETURN_CODE_TYPE &return_code);
274
275 // The Get_Bus_Status(I/O) function allows the PSSS UoC to query
276 // for status information of the bus underlying an I/O connection.
277             void Get_Bus_Status(
278 /*in*/CONNECTION_HANDLE_TYPE handle,
279 /*out*/BUS_STATUS_TYPE *status,
280 /*out*/RETURN_CODE_TYPE &return_code);
281 // The Register_Notification_Event(I/O) function is used by
282 // the PSSS UoC to register a callback function that is called by
283 // the I/O Service when an event occurs on the given connection.
284 //! Note: The io_callback parameter is semantically an in parameter
285 //! but is inout to avoid an undesirable mapping in C++.
286
287
288         }
289     }
290
291     namespace Configuration {
292         typedef int HANDLE_TYPE;
293         typedef long BUFFER_SIZE_TYPE;
294     }
295
296     typedef void *(*IO_Callback)(void *);
297
298     namespace IO_Service_FACE_Serial {
299
300         void Register_Notification_Event(
301             /*in*/FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
302             /*inout*/IO_Callback &io_callback,
303             /*out*/RETURN_CODE_TYPE &return_code);
304
305             // The Unregister_Notification_Event(I/O) function is used
306             // by the PSSS UoC to unregister the callback previously
    associated
307             // with the connection.
308         void Unregister_Notification_Event(
309             /*in*/FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
310             /*out*/RETURN_CODE_TYPE &return_code);
311             //}; // interface IO_Service
312
313     }
```

```
} // module FACE



/*#ifdef __cplusplus
extern "C"
#endif __cplusplus*/

#endif // __FACE_IOS
```

A.4.1.3 IO_utils.h

```
//
// Created by Edoardo Cavallotti on 05/06/24.
//

#ifndef SERIAL_FACE_IO_UTILS_H
#define SERIAL_FACE_IO_UTILS_H

#include "IO.h"
#include "Serial/include/serial_constant.h"

extern FACE::IOSS::CONNECTION_HANDLE_TYPE conn_idx;
extern FACE::IOSS::CONNECTION_HANDLE_TYPE valid_handles[MAX_CONNECTIONS];
extern FACE::IO_Callback callback_vector[MAX_CONNECTIONS];

bool isTimeoutOutsideRange(FACE::TIMEOUT_TYPE timeout);
bool isHandleUnknown(FACE::IOSS::CONNECTION_HANDLE_TYPE handle);
bool isHandleUnclosable(FACE::IOSS::CONNECTION_HANDLE_TYPE handle);
bool paramsMatch(FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE list1,
                 FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE list2);
int setParametersToPort();

int ReleaseHandleResources(FACE::IOSS::CONNECTION_HANDLE_TYPE handle);
FACE::IOSS::Serial::CONNECTION_STATUS_TYPE QueryConnectionStatus(
        FACE::IOSS::CONNECTION_HANDLE_TYPE handle);
FACE::IOSS::Serial::BUS_STATUS_TYPE QueryBusStatus(FACE::IOSS::
    CONNECTION_HANDLE_TYPE handle);


void Get_Connection_Status(
        FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
        FACE::IOSS::Serial::CONNECTION_STATUS_TYPE& status,
        FACE::RETURN_CODE_TYPE& return_code);

void Get_Bus_Status(
        FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
        FACE::IOSS::Serial::BUS_STATUS_TYPE& bus_status_value,
        FACE::RETURN_CODE_TYPE& return_code);

int setChannelNumber(FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
                     FACE::IOSS::Serial::IO_PARAMETER_VALUE_TYPE
    ioParameterValue);
int setSerialMode(FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
                  FACE::IOSS::Serial::IO_PARAMETER_VALUE_TYPE ioParameterValue
    );

int AssociateCallbackToHandle(FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
                              FACE::IO_Callback& io_callback);
int DisassociateCallbackFromHandle(FACE::IOSS::CONNECTION_HANDLE_TYPE handle);

#endif //SERIAL_FACE_IO_UTILS_H
```

A.4.1.4 IO_utils.cpp

```
//
// Created by Edoardo Cavallotti on 05/06/24.
//

#include "IO_utils.h"

FACE::IOSS::CONNECTION_HANDLE_TYPE conn_idx = 0;
FACE::IOSS::CONNECTION_HANDLE_TYPE valid_handles[MAX_CONNECTIONS];
FACE::IO_Callback callback_vector[MAX_CONNECTIONS];


bool isTimeoutOutsideRange(FACE::TIMEOUT_TYPE timeout) {
    if(timeout>1000000000L){
        return true;
    }
    return false;
}

bool isHandleUnknown(FACE::IOSS::CONNECTION_HANDLE_TYPE handle){
    bool flag = true;
    for(short i=0; i<MAX_CONNECTIONS; i++){
        if(valid_handles[i]==handle){
            flag=false;
        }
    }
    return flag;
}

bool isHandleUnclosable(FACE::IOSS::CONNECTION_HANDLE_TYPE handle){
    bool flag = true;
    for(short i=0; i<MAX_CONNECTIONS; i++){
        if(valid_handles[i]==handle){
            flag=false;
        }
    }
    return flag;
}

bool paramsMatch(FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE list1,
                 FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE list2){
    if(list1.items.size()!=list2.items.size()){
        return false;
    }

    for(short i=0; i<list1.items.size(); i++){
        if(list1.items.at(i).id != list2.items.at(i).id){
            return false;
        }
    }

    return true;
}

int setParametersToPort(){
    return 1;
}

int ReleaseHandleResources(FACE::IOSS::CONNECTION_HANDLE_TYPE handle){
    for(short i=0; i<MAX_CONNECTIONS; i++){
        if(valid_handles[i]==handle){
            valid_handles[i] = -1;
            callback_vector[i] = nullptr;
            break;
        }
    }

    return 0;
}

FACE::IOSS::Serial::CONNECTION_STATUS_TYPE QueryConnectionStatus(
        FACE::IOSS::CONNECTION_HANDLE_TYPE handle){
    FACE::IOSS::Serial::CONNECTION_STATUS_TYPE ris;
    return ris;
}

```

```
76 FACE::IOSS::Serial::BUS_STATUS_TYPE QueryBusStatus(
77         FACE::IOSS::CONNECTION_HANDLE_TYPE handle){
78     FACE::IOSS::Serial::BUS_STATUS_TYPE ris;
79     return ris;
80 }
81
82 void Get_Connection_Status(
83         FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
84         FACE::IOSS::Serial::CONNECTION_STATUS_TYPE& status,
85         FACE::RETURN_CODE_TYPE& return_code)
86 {
87     if (isHandleUnknown(handle)) {
88         return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
89         return;
90     }
91
92
93     status = QueryConnectionStatus(handle);
94     return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
95 }
96
97 void Get_Bus_Status(
98         FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
99         FACE::IOSS::Serial::BUS_STATUS_TYPE& bus_status_value,
100        FACE::RETURN_CODE_TYPE& return_code)
101 {
102     if (isHandleUnknown(handle)) {
103         return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
104         return;
105     }
106     bus_status_value = QueryBusStatus(handle);
107     return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
108 }
109
110
111 int setChannelNumber(FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
112                      FACE::IOSS::Serial::IO_PARAMETER_VALUE_TYPE
    ioParameterValue){
113     return 0;
114 }
115 int setSerialMode(FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
116                   FACE::IOSS::Serial::IO_PARAMETER_VALUE_TYPE ioParameterValue
    ){
117     return 0;
118 }
119
120 int AssociateCallbackToHandle(FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
121                               FACE::IO_Callback& io_callback){
122     for(short i=0; i<MAX_CONNECTIONS; i++){
123         if(valid_handles[i]==handle){
124             if(callback_vector[i]== nullptr){
125                 callback_vector[i] = io_callback;
126                 return 0;
127             }
128             else{
129                 return 1;
130             }
131         }
132     }
133     return -1;
134 }
135
136 int DisassociateCallbackFromHandle(FACE::IOSS::CONNECTION_HANDLE_TYPE handle){
137
138     for(short i=0; i<MAX_CONNECTIONS; i++){
139         if(valid_handles[i]==handle){
140             if(callback_vector[i] == nullptr){
141                 return 1;
142             }
143             else{
144                 callback_vector[i] = nullptr;
145                 return 1;
146             }
147         }
148     }
```

```
149     return -1;
150 }
```

A.4.2 Serial

A.4.2.1 ConfigFile.h

```
1  //
2  // Created by Edoardo Cavallotti on 05/06/24.
3  //
4
5  #ifndef SERIAL_CONFIGFILE_H
6  #define SERIAL_CONFIGFILE_H
7
8  #include "../../IO.h"
9  #include "unistd.h"
10 #include <string>
11
12 #define MAX_LINE_LENGTH 64
13
14 using namespace std;
15
16
17 class ConfigFile {
18 public:
19     ConfigFile();
20     virtual ~ConfigFile();
21
22     void Initialize(FACE::CONFIGURATION_RESOURCE cfg_file, FACE::
       RETURN_CODE_TYPE& return_code);
23     void Open(string fullName, int& fd, FACE::RETURN_CODE_TYPE& return_code);
24     void Read(int fd, string str, void * buffer,
25                 size_t buff_sz, FACE::Configuration::BUFFER_SIZE_TYPE sz,
26                 FACE::RETURN_CODE_TYPE& return_code);
27     void Close(int fd, FACE::RETURN_CODE_TYPE& return_code);
28 private:
29     int fd;
30 };
31
32
33 #endif //SERIAL_CONFIGFILE_H
```

A.4.2.2 MyCallback.h

```
1  //
2  // Created by Edoardo Cavallotti on 20/06/24.
3  //
4
5  #ifndef SERIAL_MYCALLBACK_H
6  #define SERIAL_MYCALLBACK_H
7
8  #include "../../IO.h"
9
10
11 class MyCallback {
12 public:
13     MyCallback(void* (*callbackFunc)(void*));
14     ~MyCallback();
15     void Process_Notification_Event(FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
        FACE::IOSS::Serial::NOTIFICATION_EVENT_TYPE event_type);
16 private:
17     void* (*cb)(void* args);
18 };
19
20
```

```
#endif //SERIAL_MYCALLBACK_H
```

A.4.2.3 Serial.h

```
//
// Created by Edoardo Cavallotti on 05/06/24.
//

#ifndef SERIAL_SERIAL_H
#define SERIAL_SERIAL_H

#include "../../IO.h"
#include <string>
#include "../../IO_utils.h"
#include "ConfigFile.h"
#include "MyCallback.h"


class FACE_Serial{
public:
    FACE_Serial(string devName);
    FACE_Serial();
    virtual ~FACE_Serial();
// Method signatures will appear here and are detailed below
    void Initialize(
            const FACE::CONFIGURATION_RESOURCE& config_resource,
            FACE::RETURN_CODE_TYPE& return_code);

    void Open_Connection(
            const FACE::IOSS::Serial::CONNECTION_NAME_TYPE& name,
            FACE::TIMEOUT_TYPE timeout,
            FACE::IOSS::CONNECTION_HANDLE_TYPE& handle,
            FACE::RETURN_CODE_TYPE& return_code);

    void Close_Connection(
            FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
            FACE::RETURN_CODE_TYPE& return_code);

    void Read(FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
            FACE::TIMEOUT_TYPE timeout,
            FACE::IOSS::Serial::READ_PAYLOAD_TYPE *rd,
            FACE::UnsignedLong effective_payload_size,
            FACE::RETURN_CODE_TYPE& return_code);

    void Write(FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
                FACE::TIMEOUT_TYPE timeout,
                FACE::IOSS::Serial::WRITE_PAYLOAD_TYPE *wr,
                FACE::UnsignedLong effective_payload_size,
                FACE::RETURN_CODE_TYPE& return_code);

    void Flush(FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
                FACE::RETURN_CODE_TYPE& return_code);

    void Configure_Connection_Parameters(
            FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
            FACE::TIMEOUT_TYPE timeout,
            FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE& parameters,
            FACE::RETURN_CODE_TYPE& return_code);

    void Get_Connection_Configuration (
            FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
            FACE::TIMEOUT_TYPE timeout,
            FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE& parameters,
            FACE::RETURN_CODE_TYPE& return_code);

    void Configure_Bus_Parameters (
            FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
            FACE::TIMEOUT_TYPE timeout,
            const FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE&
    parameters,
            FACE::RETURN_CODE_TYPE& return_code);
```

```
67
68      void Get_Bus_Configuration (
69              FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
70              FACE::TIMEOUT_TYPE timeout,
71              FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE& parameters,
72              FACE::RETURN_CODE_TYPE& return_code);
73
74      long getBaudRate() const;
75      short getDataBits() const;
76      short getStopBits() const;
77      short getParity() const;
78      short getFlowControl() const;
79      bool isEcho() const;
80      bool isTranslateLf() const;
81      bool isTranslateEchoCr() const;
82      const char *getDeviceName() const;
83
84  private:
85      long baudRate;
86      short dataBits;
87      short stopBits;
88      short parity;
89      short flowControl;
90      bool echo;
91      bool translateLF;
92      bool translateEchoCR;
93      char deviceName[MAX_LINE_LENGTH];
94      char *name;
95      int fd_m;
96
97      ConfigFile *configFile = NULL;
98      MyCallback *my_cb = NULL;
99  };
100
101
102 #endif //SERIAL_SERIAL_H
```

A.4.2.4 serial_constant.h

```
1  //
2  // Created by Edoardo Cavallotti on 05/06/24.
3  //
4
5  #ifndef SERIAL_SERIAL_CONSTANT_H
6  #define SERIAL_SERIAL_CONSTANT_H
7
8  /*
9   * SERIAL CONSTANT DEFINITION
10  */
11
12 #define FACE_IOSS_Serial_CHANNEL_NUM 0
13
14 #define FACE_IOSS_Serial_FLOW_CONTROL 1
15 #define FACE_IOSS_Serial_BAUD_RATE 2
16 #define FACE_IOSS_Serial_MODE 3
17 #define FACE_IOSS_Serial_RS_232 4
18 #define FACE_IOSS_Serial_RS_422 5
19 #define FACE_IOSS_Serial_RS_485 6
20
21
22 #define FACE_IOSS_Serial_NONE 7
23 #define FACE_IOSS_Serial_XON_XOFF 8
24 #define FACE_IOSS_Serial_RTS_CTS 9
25 #define FACE_IOSS_Serial_DSR_DTR 10
26
27
28 #define FACE_IOSS_Serial_PARITY 11
29 #define FACE_IOSS_Serial_PARITY_NONE 12
30 #define FACE_IOSS_Serial_PARITY_ODD 13
31 #define FACE_IOSS_Serial_PARITY_EVEN 14
32 #define FACE_IOSS_Serial_PARITY_MARK 15
```

```
#define FACE_IOSS_Serial_PARITY_SPACE 16

#define FACE_IOSS_Serial_DATA_BITS 17
#define FACE_IOSS_Serial_STOP_BITS 18

#endif //SERIAL_SERIAL_CONSTANT_H
```

A.4.2.5 ConfigFile.cpp

```
//
// Created by Edoardo Cavallotti on 05/06/24.
//

#include "include/ConfigFile.h"

void ConfigFile::Initialize(FACE::CONFIGURATION_RESOURCE cfg_file, FACE::
    RETURN_CODE_TYPE& return_code){
    return_code = FACE::NO_ERROR;
}

void ConfigFile::Open(string fullName, int& fd, FACE::RETURN_CODE_TYPE&
    return_code){

   fd = open(fullName.c_str(), O_RDONLY, 0x777);

    if(fd==-1){
        printf("Opening Error: %s\n", strerror(errno));
        return_code = FACE::INVALID_CONFIG;
        return;
    }
    return_code = FACE::NO_ERROR;
}

void ConfigFile::Read(int fd, string str, void *buffer,
          size_t buff_sz, FACE::Configuration::BUFFER_SIZE_TYPE sz,
          FACE::RETURN_CODE_TYPE& return_code){

    if (str.length() > buff_sz) {
        return_code = FACE::DATA_BUFFER_TOO_SMALL;
        return;
    }

    if (fd == -1) {
        return_code = FACE::INVALID_CONFIG;
        return;
    }

    ssize_t bytesRead;
    size_t totalBytesRead = 0;
    bool found = false;

    lseek(fd, 0, SEEK_SET); //rewind

    char c;
    int len = 0;
    char linea[MAX_LINE_LENGTH];
    char tag[MAX_LINE_LENGTH];
    char *value = (char*) buffer;

    while(read(fd, (void*) &c, sizeof(c))>0){
        if(c=='\n'){
            linea[len] = '\0';
            //printf("%s (%d)\n",linea, len);
            int j = 0;
            while(linea[j]!='='){
                tag[j] = linea[j];
                j++;
            }

            tag[j] = '\0';

```

```
            if(strcmp(tag, str.c_str()) == 0){
                //printf("%s, %s\n", tag, str.c_str());
                for(short k=0; k<(len-j); k++){
                    value[k] = linea[j+k+1];
                }
                value[len-j] = '\0';
                //printf("%s\n", value);
                break;
            }

            len = 0;
        }
        else{
            linea[len] = c;
            //printf("%c", c);
            len++;
        }
    }

    buffer = (void*) value;
    return_code = FACE::NO_ERROR;  // Non trovato ma nessun errore

}


void ConfigFile::Close(int fd, FACE::RETURN_CODE_TYPE& return_code){
    if(fd==-1){
        return_code = FACE::INVALID_CONFIG;
        return;
    }
    if(!close(fd)){
        return_code = FACE::INVALID_CONFIG;
        return;
    }
    return_code = FACE::NO_ERROR;
}

ConfigFile::ConfigFile() {

}

ConfigFile::~ConfigFile() {

}
```

A.4.2.6 MyCallback.cpp

```
//
// Created by Edoardo Cavallotti on 20/06/24.
//

#include "include/MyCallback.h"

MyCallback::MyCallback(void* (*callbackFunc)(void*)){
    this->cb = callbackFunc;
}
MyCallback::~MyCallback(){
}

void MyCallback::Process_Notification_Event(FACE::IOSS::CONNECTION_HANDLE_TYPE
     handle, FACE::IOSS::Serial::NOTIFICATION_EVENT_TYPE event_type){
    switch(event_type){
        case FACE::IOSS::Serial::DATA_READ_EVENT:
            break;
        case FACE::IOSS::Serial::CONNECTION_CONFIG_CHANGE_EVENT:
            break;
        case FACE::IOSS::Serial::CONNECTION_STATUS_CHANGE_EVENT:
            break;
        case FACE::IOSS::Serial::BUS_CONFIG_CHANGE_EVENT:
            break;
        case FACE::IOSS::Serial::BUS_STATUS_CHANGE_EVENT:
```

```
            break;
        }
}
```

A.4.2.7 Serial.cpp

```
//
// Created by Edoardo Cavallotti on 05/06/24.
//

#include "include/Serial.h"
#include <termios.h>

// hidden function
speed_t convertBaudRate(int baudRate, FACE::RETURN_CODE_TYPE& return_status) {
    return_status = FACE::NO_ERROR;
    switch(baudRate) {
        case 50: return B50;
        case 75: return B75;
        case 110: return B110;
        case 134: return B134;
        case 150: return B150;
        case 200: return B200;
        case 300: return B300;
        case 600: return B600;
        case 1200: return B1200;
        case 1800: return B1800;
        case 2400: return B2400;
        case 4800: return B4800;
        case 9600: return B9600;
        case 19200: return B19200;
        case 38400: return B38400;
        case 57600: return B57600;
        case 115200: return B115200;
        case 230400: return B230400;
        default: {
            return_status = FACE::INVALID_PARAM;
            return 0;
        }
    }
}

int convertDataBit(int dataBits, FACE::RETURN_CODE_TYPE& return_status){
    return_status = FACE::NO_ERROR;
    int ris;
    switch(dataBits){
        case 5:{
            ris = CS5;
            break;
        }
        case 6:{
            ris = CS6;
            break;
        }
        case 7:{
            ris = CS7;
            break;
        }
        case 8:{
            ris = CS8;
            break;
        }
        default:{
            ris = -1;
            return_status = FACE::INVALID_PARAM;
            break;
        }

    }
    return ris;
}
```

```
66
67
68  FACE_Serial::FACE_Serial(string devName) {
69      configFile = new ConfigFile();
70      my_cb = new MyCallback(nullptr);
71      name = (char*) malloc(16*sizeof(char));
72      strcpy(name, "config.cfg");
73      stpcpy(deviceName, devName.c_str());
74  }
75
76  FACE_Serial::FACE_Serial() {
77      configFile = new ConfigFile();
78      my_cb = new MyCallback(nullptr);
79      name = (char*) malloc(16*sizeof(char));
80      strcpy(name, "config.cfg");
81  }
82
83  void FACE_Serial::Initialize(
84          const FACE::CONFIGURATION_RESOURCE& config_resource,
85          FACE::RETURN_CODE_TYPE& return_code) {
86  // Assume safe values for class member variables are assignd in the
87  // C++ constructor. Assign practical defaults here before reading
88  // configured values.
89      baudRate=115200; dataBits=8; stopBits=1; parity=0; flowControl=0;
90      echo=translateLF=translateEchoCR=false;
91      fd_m = -1;
92      for(short i=0; i<MAX_CONNECTIONS; i++){
93          callback_vector[i] = nullptr;
94      }
95
96  // Check for NO_ACTION, NOT_AVAILABLE, and IN_PROGRESS error
97  // conditions.
98  // Use Configuration Services to read configured values from
99  //     config_resource     . Perform validity checks on each value before
100 // assignment when applicable, returning INVALID_CONFIG if check fails.
101     if(configFile != NULL){
102         FACE::Configuration::HANDLE_TYPE mySection;
103         std::string fullName(config_resource);
104         fullName.append("_"); fullName.append(name);
105         configFile->Open(fullName, mySection, return_code);
106         if(return_code == FACE::NO_ERROR){
107             char buffer[129]; // assume no config line is larger than 128
108             FACE::Configuration::BUFFER_SIZE_TYPE sz = 128;
109             configFile->Read(mySection,"deviceName",static_cast<void *>(buffer
    ),
110                              sizeof(buffer), sz, return_code);
111             if(return_code==FACE::NO_ERROR)
112                 strncpy(deviceName,buffer,sizeof(deviceName)-1);
113             deviceName[sizeof(deviceName)-1]=0;
114             configFile->Read(mySection,"baudRate",static_cast<void *>(buffer),
115                              sizeof(buffer),sz,return_code);
116             int brate=atoi(buffer);
117             if(brate >=50 && brate <=230400) baudRate=brate;
118             configFile->Read(mySection,"dataBits",static_cast<void *>(buffer),
119                              sizeof(buffer),sz,return_code);
120             int databits=atoi(buffer);
121             if(databits==7 || databits==8 ) dataBits=databits;
122             configFile->Read(mySection,"stopBits",static_cast<void *>(buffer),
123                              sizeof(buffer),sz,return_code);
124             int stopbits=atoi(buffer);
125             if(stopbits==1 || stopbits==2 ) stopBits=stopbits;
126             configFile->Read(mySection,"parity",static_cast<void *>(buffer),
127                              sizeof(buffer),sz,return_code);
128             int parbits=atoi(buffer);
129             if(parbits>=0 && parbits<=2 ) parity=parbits;
130             configFile->Read(mySection,"flowControl",static_cast<void *>(
    buffer),
131                              sizeof(buffer),sz,return_code);
132             int flowbits=atoi(buffer);
133             if(flowbits>=0 && flowbits<=2 ) flowControl=flowbits;
134             configFile->Read(mySection,"echo",static_cast<void *>(buffer),
135                              sizeof(buffer),sz,return_code);
136             int echobits=atoi(buffer);
137             if(echobits==0 || echobits==1 ) echo=(echobits==1);
```

```
138             configFile->Read(mySection,"translateLF",static_cast<void *>(
    buffer),
139                              sizeof(buffer),sz,return_code);
140             int lfbits=atoi(buffer);
141             if(lfbits==0 || lfbits==1 ) translateLF=(lfbits==1);
142             configFile->Read(mySection,"translateEchoCR",static_cast<void *>(
    buffer),
143                              sizeof(buffer),sz,return_code);
144             int crbits=atoi(buffer);
145             if(crbits==0 || crbits==1 ) translateEchoCR=(crbits==1);
146             configFile->Close(mySection, return_code);
147             return_code=FACE::NO_ERROR;
148         } else fprintf(stderr,
149                        "FACE_Serial failed to open section: %s. return_code=%d
    \n",
150                        name, return_code);
151     }
152     else{
153         printf("Config File null");
154         return_code=FACE::INVALID_CONFIG;
155     }
156 }
157
158 void FACE_Serial::Open_Connection(
159         const FACE::IOSS::Serial::CONNECTION_NAME_TYPE& name,
160         FACE::TIMEOUT_TYPE timeout,
161         FACE::IOSS::CONNECTION_HANDLE_TYPE& handle,
162         FACE::RETURN_CODE_TYPE& return_code) {
163     if (isTimeoutOutsideRange(timeout)) {
164         return_code = FACE::INVALID_PARAM;
165         return;
166     }
167     fd_m = open(deviceName, O_RDWR | O_NOCTTY | O_SYNC);
168     if (fd_m < 0) {
169         fprintf(stderr,"FACE_Serial failed to open serial device: %s. errno=%d
    \n",
170                 deviceName,errno);
171         return_code=FACE::INVALID_CONFIG;
172     } else {
173         fprintf(stderr,"(NOERR) DEBUG: Serial port %s opened at speed_NED %ld\
    n",deviceName, baudRate);
174
175         struct termios options;
176         memset(&options, 0, sizeof(options));
177
178         // Ottenere i parametri correnti della porta seriale
179         if (tcgetattr(fd_m, &options) != 0) {
180             perror("Error in reading!!");
181             close(fd_m);
182             return_code = FACE::INVALID_CONFIG;
183             return;
184         }
185
186         //this->baudRate
187
188         // set baud rate
189         //printf("BR: %ld\n", baudRate);
190         cfsetispeed(&options, convertBaudRate(baudRate, return_code));
191         cfsetospeed(&options, convertBaudRate(baudRate, return_code));
192
193         if(return_code!=FACE::NO_ERROR){
194             perror("Baud Rate invalid!!");
195         }
196
197         // Impostare 8 bit di dati, 1 bit di stop, no parity
198         options.c_cflag &= ~PARENB;     // disable parity
199         options.c_cflag &= ~CSTOPB;     // 1 bit stop
200         options.c_cflag &= ~CSIZE;
201         options.c_cflag |= convertDataBit(dataBits, return_code);
202
203         if(return_code!=FACE::NO_ERROR){
204             perror("Data Bits invalid!!");
205         }
206
207         // set timeout
```

```
208         /*options.c_cc[VMIN] = 0;
209         if(timeout==FACE::IOSS::Serial::WAIT_FOREVER){ //blocking
210             options.c_cc[VTIME] = 0;
211             options.c_cc[VMIN] = 1;
212         }
213         else if(timeout==FACE::IOSS::Serial::NO_WAIT){
214             options.c_cc[VTIME] = 0;
215         }
216         else{
217             options.c_cc[VTIME] = timeout/1e8; // from nsec to dsec
218         }*/
219 
220         // set params to serial port
221         if (tcsetattr(fd_m, TCSANOW, &options) != 0) {
222             perror("Error in configuration!!");
223             close(fd_m);
224             return_code = FACE::INVALID_CONFIG;
225             return;
226         }
227 
228         handle = fd_m;
229         valid_handles[conn_idx] = handle;
230         conn_idx++;
231         conn_idx%=MAX_CONNECTIONS;
232     }
233 
234     if (!setParametersToPort()){ // Helper method to hide details
235         close(fd_m);
236         fd_m=-1;
237         return_code=FACE::INVALID_CONFIG;
238     }
239     fflush(stdout);
240     return_code=FACE::NO_ERROR;
241 
242 }
243 
244 void FACE_Serial::Close_Connection(
245         FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
246         FACE::RETURN_CODE_TYPE& return_code){
247     if (isHandleUnknown(handle)) {
248         return_code = FACE::INVALID_PARAM;
249         return;
250     }
251     if (isHandleUnclosable(handle)) {
252         return_code = FACE::INVALID_MODE;
253         return;
254     }
255     if (ReleaseHandleResources(handle)) {
256         return_code = FACE::NO_ERROR;
257         close(fd_m);
258         fd_m = -1;
259         return;
260     } else {
261         return_code = FACE::NO_ACTION;
262         close(fd_m);
263         fd_m = -1;
264         return;
265     }
266 }
267 
268 void FACE_Serial::Read(
269         FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
270         FACE::TIMEOUT_TYPE timeout,
271         FACE::IOSS::Serial::READ_PAYLOAD_TYPE *rd,
272         FACE::UnsignedLong effective_payload_size,
273         FACE::RETURN_CODE_TYPE& return_code){
274 
275     return_code = FACE::NO_ERROR;
276 
277     if(isTimeoutOutsideRange(timeout)){
278         return_code = FACE::INVALID_PARAM;
279         return;
280     }
281     if(isHandleUnknown(handle)){
282         return_code = FACE::INVALID_PARAM;
```

```
283         return;
284     }
285     if(effective_payload_size>PAYLOAD_DATA_MSG_SIZE){
286         return_code = FACE::DATA_BUFFER_TOO_SMALL;
287         return;
288     }
289 
290     struct termios options;
291     memset(&options, 0, sizeof(options));
292 
293     // Ottenere i parametri correnti della porta seriale
294     if (tcgetattr(handle, &options) != 0) {
295         perror("Error in reading!!");
296         close(handle);
297         return_code = FACE::INVALID_CONFIG;
298         return;
299     }
300 
301     options.c_cc[VMIN] = 0;
302     if(timeout==FACE::IOSS::Serial::WAIT_FOREVER or timeout==FACE::
    INF_TIME_VALUE){ //blocking
303         options.c_cc[VTIME] = 0;
304         options.c_cc[VMIN] = 1;
305     }
306     else if(timeout==FACE::IOSS::Serial::NO_WAIT){
307         options.c_cc[VTIME] = 0;
308     }
309     else{
310         options.c_cc[VTIME] = timeout/1e8; // from nsec to dsec
311     }
312 
313     // set params to serial port
314     if (tcsetattr(handle, TCSANOW, &options) != 0) {
315         perror("Error in configuration!!");
316         close(handle);
317         return_code = FACE::INVALID_CONFIG;
318         return;
319     }
320 
321     ssize_t total_bytes_read = 0;
322     ssize_t bytes_read;
323 
324     if(effective_payload_size>0){
325         bytes_read = read(handle, rd->payload, effective_payload_size);
326 
327         if(bytes_read>0 && bytes_read<effective_payload_size){
328             return_code = FACE::INVALID_CONFIG;
329             return;
330         }
331 
332         if(bytes_read==0){
333             return_code = FACE::TIMED_OUT;
334             return;
335         }
336 
337         if(bytes_read==effective_payload_size){
338             struct timespec ts;
339             clock_gettime(CLOCK_MONOTONIC, &ts);
340             rd->received_time = (long long) (ts.tv_nsec)+(ts.tv_sec*1000000000
    LL);
341             return_code = FACE::NO_ERROR;
342             return;
343         }
344     }
345     else{
346         char buffer[DATA_CHUNK_SIZE];
347 
348         while(1){
349             bytes_read = read(handle, buffer, DATA_CHUNK_SIZE);
350 
351             if(bytes_read==0 and total_bytes_read==0){
352                 return_code = FACE::TIMED_OUT;
353                 return;
354             }
355 
```

```
356             if(bytes_read<DATA_CHUNK_SIZE){
357                 memcpy(rd->payload + total_bytes_read, buffer, bytes_read);
358                 total_bytes_read+=bytes_read;
359                 return_code = FACE::NO_ERROR;
360                 struct timespec ts;
361                 clock_gettime(CLOCK_MONOTONIC, &ts);
362                 rd->received_time = (long long) (ts.tv_nsec)+(ts.tv_sec
    *1000000000LL);
363                 return;
364             }
365 
366             if(bytes_read==DATA_CHUNK_SIZE){
367                 memcpy(rd->payload + total_bytes_read, buffer, bytes_read);
368                 total_bytes_read+=bytes_read;
369             }
370         }
371 
372     }
373 
374 }
375 
376 void FACE_Serial::Write(FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
377                         FACE::TIMEOUT_TYPE timeout,
378                         FACE::IOSS::Serial::WRITE_PAYLOAD_TYPE *wr,
379                         FACE::UnsignedLong effective_payload_size,
380                         FACE::RETURN_CODE_TYPE& return_code){
381 
382     return_code = FACE::NO_ERROR;
383     if(isTimeoutOutsideRange(timeout)){
384         return_code = FACE::INVALID_PARAM;
385         printf("Error: Timeout outside range!!\n");
386         return;
387     }
388     if(isHandleUnknown(handle)){
389         return_code = FACE::INVALID_PARAM;
390         printf("Error: Unknown Handle (%lld)!!\n", handle);
391         return;
392     }
393     if(effective_payload_size>PAYLOAD_DATA_MSG_SIZE){
394         return_code = FACE::DATA_BUFFER_TOO_SMALL;
395         return;
396     }
397 
398     struct termios options;
399     memset(&options, 0, sizeof(options));
400 
401     // Retrieve current parameters for serial port
402     if (tcgetattr(handle, &options) != 0) {
403         perror("Error in reading!!");
404         close(handle);
405         return_code = FACE::INVALID_CONFIG;
406         return;
407     }
408 
409     options.c_cc[VMIN] = 0;
410     if(timeout==FACE::IOSS::Serial::WAIT_FOREVER){ //blocking
411         options.c_cc[VTIME] = 0;
412         options.c_cc[VMIN] = 1;
413     }
414     else if(timeout==FACE::IOSS::Serial::NO_WAIT){
415         options.c_cc[VTIME] = 0;
416     }
417     else{
418         options.c_cc[VTIME] = timeout/1e8; // from nsec to dsec
419     }
420 
421     // set params to serial port
422     if (tcsetattr(handle, TCSANOW, &options) != 0) {
423         perror("Error in configuration!!");
424         close(handle);
425         return_code = FACE::INVALID_CONFIG;
426         return;
427     }
428 
429     ssize_t bytes_written = 0;
```

```
430
431     if (effective_payload_size == 0) {
432         bytes_written = write(handle, wr->payload, PAYLOAD_DATA_MSG_SIZE);
433         fsync(handle);
434     } else {
435         bytes_written = write(handle, wr->payload, (size_t)
    effective_payload_size);
436         fsync(handle);
437     }
438
439     if(bytes_written<=0){
440         if (errno == EAGAIN || errno == EWOULDBLOCK) {
441             return_code = FACE::NOT_AVAILABLE;
442         } else {
443             return_code = FACE::INVALID_CONFIG;
444             perror("Serial write");
445         }
446     }
447     else{
448         return_code = FACE::NO_ERROR;
449     }
450 }
451
452 void FACE_Serial::Flush(FACE::IOSS::CONNECTION_HANDLE_TYPE handle, FACE::
    RETURN_CODE_TYPE& return_code){
453     if (tcflush(handle, TCIOFLUSH) != 0) {
454         perror("tcflush error...\n");
455         return_code = FACE::INVALID_CONFIG;
456     }
457     else{
458         return_code = FACE::NO_ERROR;
459     }
460
461 }
462
463
464 void FACE_Serial::Configure_Connection_Parameters(
465         FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
466         FACE::TIMEOUT_TYPE timeout,
467         FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE& parameters,
468         FACE::RETURN_CODE_TYPE& return_code) {
469     FACE::UnsignedLong i;
470     FACE::IOSS::IO_PARAMETER_ID_TYPE ioParameterId;
471     FACE::IOSS::Serial::IO_PARAMETER_VALUE_TYPE ioParameterValue;
472     FACE::UnsignedLong numberOfParametersToSet;
473     if (isHandleUnknown(handle)) {
474         return_code = FACE::INVALID_PARAM;
475         return;
476     }
477     if (isTimeoutOutsideRange(timeout)) {
478         return_code = FACE::INVALID_PARAM;
479         return;
480     }
481     if ((numberOfParametersToSet = parameters.items.size()) < 1) {
482 // No parameters to be set
483         return_code = FACE::NO_ERROR;
484         return;
485     }
486 // Loop over each item in parameters to verify that each item has a
487 // recognized ID and a legal value
488     for (i=0; i<numberOfParametersToSet; i++) {
489         ioParameterId = parameters.items[i].id;
490         ioParameterValue = parameters.items[i].value;
491         switch (ioParameterId) {
492             case FACE_IOSS_Serial_CHANNEL_NUM:
493                 // If ioParameterValue is not a legal value for the channel
494                 // - set return_code to INVALID_PARAM
495                 // - return
496                 if(ioParameterValue.discriminator!=FACE::IOSS::Serial::
    IO_PARAMETER_VALUE_TYPES_TYPE::FACE_USHORT){
497                     return_code = FACE::INVALID_PARAM;
498                 }
499                 break;
500             case FACE_IOSS_Serial_MODE:
```

```
501                 // If ioParameterValue is not a legal value for the the serial
     mode
502                 // - set return_code to INVALID_PARAM
503                 // - return
504                 if(ioParameterValue.discriminator!=FACE::IOSS::Serial::
    IO_PARAMETER_VALUE_TYPES_TYPE::FACE_USHORT){
505                     return_code = FACE::INVALID_PARAM;
506                 }
507                 break;
508 // . . .
509             default:
510 // Unrecognized parameter ID
511                 return_code = FACE::INVALID_PARAM;
512                 return;
513         }
514     }
515 // Loop over each item in parameters to set each parameter
516     // Loop over each item in parameters to set each parameter
517     for (i = 0; i < numberOfParametersToSet; i++) {
518         ioParameterId = parameters.items[i].id;
519         ioParameterValue = parameters.items[i].value;
520 
521         switch (ioParameterId) {
522             case FACE_IOSS_Serial_CHANNEL_NUM:
523                 // Set the channel number to ioParameterValue
524                 if (ioParameterValue.discriminator == FACE::IOSS::Serial::
    IO_PARAMETER_VALUE_TYPES_TYPE::FACE_USHORT) {
525                     setChannelNumber(handle, ioParameterValue); // Example
    function, adapt as needed
526                 }
527                 break;
528 
529             case FACE_IOSS_Serial_MODE:
530                 // Set the serial mode to ioParameterValue
531                 if (ioParameterValue.discriminator == FACE::IOSS::Serial::
    IO_PARAMETER_VALUE_TYPES_TYPE::FACE_USHORT) {
532                     setSerialMode(handle, ioParameterValue); // Example
    function, adapt as needed
533                 }
534                 break;
535 
536                 // Add additional cases for other parameter IDs as needed
537                 // . . .
538         }
539     }
540 
541 
542     my_cb->Process_Notification_Event(handle,FACE::IOSS::Serial::
    NOTIFICATION_EVENT_TYPE::CONNECTION_CONFIG_CHANGE_EVENT);
543     return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
544 }
545 
546 void FACE_Serial::Get_Connection_Configuration (
547         FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
548         FACE::TIMEOUT_TYPE timeout,
549         FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE& parameters,
550         FACE::RETURN_CODE_TYPE& return_code) {
551     FACE::UnsignedLong i;
552     FACE::IOSS::IO_PARAMETER_ID_TYPE ioParameterId;
553     FACE::UnsignedLong numberOfParametersToGet;
554     if (isHandleUnknown(handle)) {
555         return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
556         return;
557     }
558 
559     if (isTimeoutOutsideRange(timeout)) {
560         return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
561         return;
562     }
563     if ((numberOfParametersToGet = parameters.items.size()) < 1) {
564 // No parameters to get
565         return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
566         return;
567     }
568 // Loop over each item in parameters
```

```
569     for (i=0; i<numberOfParametersToGet; i++) {
570         ioParameterId = parameters.items[i].id;
571         switch (ioParameterId) {
572             case FACE_IOSS_Serial_CHANNEL_NUM:
573 // Set parameters.items[i].value to the current channel number
574                 parameters.items[i].value.discriminator = FACE::IOSS::Serial::
    IO_PARAMETER_VALUE_TYPES_TYPE::FACE_USHORT;
575                 parameters.items[i].value.values.long_value = 10;
576                 break;
577             case FACE_IOSS_Serial_MODE:
578 // Set parameters.items[i].value to the current serial mode
579                 parameters.items[i].value.discriminator = FACE::IOSS::Serial::
    IO_PARAMETER_VALUE_TYPES_TYPE::FACE_USHORT;
580                 parameters.items[i].value.values.long_value = 10;
581                 break;
582 // . . .
583             default:
584 // Unrecognized parameter ID
585                 return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
586                 return;
587         }
588     }
589     return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
590 }
591
592 void FACE_Serial::Configure_Bus_Parameters (
593         FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
594         FACE::TIMEOUT_TYPE timeout,
595         const FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE& parameters,
596         FACE::RETURN_CODE_TYPE& return_code)
597 {
598     if (isHandleUnknown(handle)) {
599         return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
600         return;
601     }
602
603     if (isTimeoutOutsideRange(timeout)) {
604         return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
605         return;
606     }
607
608     FACE::UnsignedLong i;
609     FACE::IOSS::IO_PARAMETER_ID_TYPE ioParameterId;
610     FACE::IOSS::Serial::IO_PARAMETER_VALUE_TYPE ioParameterValue;
611     FACE::UnsignedLong numberOfParametersToSet = parameters.items.size();
612     unsigned int local_config_variable; // Assume that configuration data is
    packed in a single integer
613
614     FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE original_config_params,
    new_config_params;
615     FACE::IOSS::BUS_STATUS_TYPE original_bus_status, new_bus_status;
616     FACE::IOSS::Serial::CONNECTION_STATUS_TYPE original_connect, new_connect;
617     FACE::RETURN_CODE_TYPE config_return_code, bus_status_return_code,
    status_return_code;
618     if (numberOfParametersToSet < 1) {
619         // No parameters to be set
620         return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
621         return;
622     }
623
624     // The I/O Service processes all I/O parameters as one transaction that
625     // succeeds or fails. If the transaction fails, the I/O parameter values
    are
626     // unchanged upon return.
627     Get_Connection_Configuration (handle, timeout, original_config_params,
628                                   config_return_code);
629     Get_Bus_Status (handle, original_bus_status, bus_status_return_code);
630     Get_Connection_Status (handle, original_connect, status_return_code);
631
632     // Loop over each item in parameters
633     for (i=0; i<numberOfParametersToSet; i++) {
634         ioParameterId = parameters.items[i].id;
635         ioParameterValue = parameters.items[i].value;
636         switch (ioParameterId) {
637
```

```
638             case FACE_IOSS_Serial_MODE: {
639                 // If ioParameterValue is a legal value for the serial port,
    set mode
640                 // in local_config_variable to that value. Otherwise, set
    return_code
641                 // to INVALID_PARAM and return.
642                 if (ioParameterValue.discriminator != FACE::IOSS::Serial::
    IO_PARAMETER_VALUE_TYPES_TYPE::FACE_USHORT) {
643                     return_code = FACE::INVALID_PARAM;
644                 } else {
645                     local_config_variable = ioParameterValue.values.
    ushort_value;
646                     return_code = FACE::NO_ERROR;
647                 }
648
649                 break;
650
651             }
652
653             case FACE_IOSS_Serial_BAUD_RATE: {
654                 // If ioParameterValue is a legal value for the serial port,
    set the
655                 // Baud Rate in local_config_variable to that value. Otherwise
    , set
656                 // return_code to INVALID_PARAM and return.
657
658                 if (ioParameterValue.discriminator != FACE::IOSS::Serial::
    IO_PARAMETER_VALUE_TYPES_TYPE::FACE_LONG) {
659                     return_code = FACE::INVALID_PARAM;
660                 } else {
661                     this->baudRate = ioParameterValue.values.long_value;
662                     local_config_variable = ioParameterValue.values.long_value
    ;
663                     return_code = FACE::NO_ERROR;
664                 }
665                 break;
666             }
667
668             case FACE_IOSS_Serial_DATA_BITS: {
669                 if (ioParameterValue.discriminator != FACE::IOSS::Serial::
    IO_PARAMETER_VALUE_TYPES_TYPE::FACE_SHORT) {
670                     return_code = FACE::INVALID_PARAM;
671                 } else {
672                     this->dataBits = ioParameterValue.values.short_value;
673                     return_code = FACE::NO_ERROR;
674                 }
675             }
676
677             case FACE_IOSS_Serial_STOP_BITS: {
678                 if (ioParameterValue.discriminator != FACE::IOSS::Serial::
    IO_PARAMETER_VALUE_TYPES_TYPE::FACE_SHORT) {
679                     return_code = FACE::INVALID_PARAM;
680                 } else {
681                     this->stopBits = ioParameterValue.values.short_value;
682                     return_code = FACE::NO_ERROR;
683                 }
684             }
685
686             case FACE_IOSS_Serial_PARITY: {
687                 if (ioParameterValue.discriminator != FACE::IOSS::Serial::
    IO_PARAMETER_VALUE_TYPES_TYPE::FACE_SHORT) {
688                     return_code = FACE::INVALID_PARAM;
689                 } else {
690                     this->parity = ioParameterValue.values.short_value;
691                     return_code = FACE::NO_ERROR;
692                 }
693             }
694
695             case FACE_IOSS_Serial_FLOW_CONTROL: {
696                 // If ioParameterValue is a legal value for the serial port,
    set the
697                 // Flow Control in local_config_variable to that value.
    Otherwise, set
698                 // return_code to INVALID_PARAM and return.
```

```
699                 if (ioParameterValue.discriminator != FACE::IOSS::Serial::
    IO_PARAMETER_VALUE_TYPES_TYPE::FACE_SHORT) {
700                     return_code = FACE::INVALID_PARAM;
701                 } else {
702                     this->flowControl = ioParameterValue.values.short_value;
703                     local_config_variable = ioParameterValue.values.
    short_value;
704                     return_code = FACE::NO_ERROR;
705                 }
706                 break;
707             }
708 
709             default: {
710                 // Unrecognized parameter ID
711                 return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
712                 return;
713             }
714         }
715     }
716 
717 // Write local_config_variable to serial port
718     if (original_connect.connection_status !=
719         FACE::IOSS::Serial::IO_CONNECTION_STATUS_TYPE::NOT_OPEN) {
720 // Dispatch BUS_CONFIG_CHANGE_EVENT to all with a registered notification
721 // callback
722     //TODO!!!
723         my_cb->Process_Notification_Event(handle, FACE::IOSS::Serial::
    NOTIFICATION_EVENT_TYPE::BUS_CONFIG_CHANGE_EVENT);
724     }
725     Get_Bus_Status (handle, original_bus_status, status_return_code );
726     if (original_bus_status != new_bus_status) {
727 // Dispatch BUS_STATUS_CHANGE_EVENT to all with a registered notification
728 // callback
729  //TODO!!!
730         my_cb->Process_Notification_Event(handle, FACE::IOSS::Serial::
    NOTIFICATION_EVENT_TYPE::BUS_STATUS_CHANGE_EVENT);
731     }
732     Get_Connection_Configuration (handle, timeout, new_config_params,
733                                   config_return_code);
734 // Compare connection parameters to determine if they changed.
735     if (!paramsMatch(original_config_params, new_config_params)) {
736 // Dispatch CONNECTION_CONFIG_CHANGE_EVENT to all with a registered
737 // notification callback
738         //TODO!!!
739         my_cb->Process_Notification_Event(handle, FACE::IOSS::Serial::
    NOTIFICATION_EVENT_TYPE::CONNECTION_CONFIG_CHANGE_EVENT);
740     }
741     Get_Connection_Status (handle, new_connect, status_return_code );
742 // Compare connection status to determine if this changed.
743     if (original_connect.connection_status != new_connect.connection_status) {
744 // Dispatch CONNECTION_STATUS_CHANGE_EVENT to all with a registered
745 // notification callback
746         // TODO
747         my_cb->Process_Notification_Event(handle, FACE::IOSS::Serial::
    NOTIFICATION_EVENT_TYPE::CONNECTION_STATUS_CHANGE_EVENT);
748     }
749 
750     return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
751 }
752 
753 
754 // The Get_Bus_Configuration() function below returns the bus configuration
755 // parameters based on the parameter IDs input into the routine.
756 void FACE_Serial::Get_Bus_Configuration (
757         FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
758         FACE::TIMEOUT_TYPE timeout,
759         FACE::IOSS::Serial::IO_PARAMETER_TRANSACTION_TYPE& parameters,
760         FACE::RETURN_CODE_TYPE& return_code)
761 {
762     FACE::UnsignedLong i;
763     //Assume that configuration data is packed in a single integer
764     unsigned int local_config_variable;
765     short Serial_Mode;
766     unsigned short Flow_Control;
767     unsigned short Parity;
```

```
768     FACE::IOSS::IO_PARAMETER_ID_TYPE ioParameterId;
769     FACE::UnsignedLong numberOfParametersToSet = parameters.items.size();
770     if (isHandleUnknown(handle)) {
771         return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
772         return;
773     }
774 
775     if (isTimeoutOutsideRange(timeout)) {
776         return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
777         return;
778     }
779     //Read local_config_variable from the serial port specified by the handle
780     //If unable to read the data within the specified timeout
781     // - set return_code to TIMED_OUT
782     // - return
783     for (i=0; i<numberOfParametersToSet; i++) {
784         ioParameterId = parameters.items[i].id;
785         switch (ioParameterId) {
786             case FACE_IOSS_Serial_MODE: {
787                 //Serial_Mode = Mask local_config_variable Operational Mode
    bits
788                 if ((Serial_Mode == FACE_IOSS_Serial_RS_232) ||
789                     (Serial_Mode == FACE_IOSS_Serial_RS_422) ||
790                     (Serial_Mode == FACE_IOSS_Serial_RS_485)) {
791                     parameters.items[i].value.discriminator = FACE::IOSS::
    Serial::FACE_USHORT;
792                     parameters.items[i].value.values.ushort_value =
    Serial_Mode;
793                 } else {
794                     return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
795                     return;
796                 }
797                 break;
798             }
799             case FACE_IOSS_Serial_FLOW_CONTROL: {
800                 //Flow_Control = Mask local_config_variable Flow Control bits
801                 if ((Flow_Control == FACE_IOSS_Serial_NONE) ||
802                     (Flow_Control == FACE_IOSS_Serial_XON_XOFF) ||
803                     (Flow_Control == FACE_IOSS_Serial_RTS_CTS) ||
804                     (Flow_Control == FACE_IOSS_Serial_DSR_DTR)) {
805                     parameters.items[i].value.discriminator = FACE::IOSS::
    Serial::FACE_USHORT;
806                     parameters.items[i].value.values.ushort_value =
    Flow_Control;
807                 } else {
808                     return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
809                     return;
810                 }
811                 break;
812             }
813             case FACE_IOSS_Serial_PARITY: {
814                 //Parity = Mask local_config_variable Parity bits
815                 if ((Parity == FACE_IOSS_Serial_PARITY_NONE) ||
816                     (Parity == FACE_IOSS_Serial_PARITY_ODD) ||
817                     (Parity == FACE_IOSS_Serial_PARITY_EVEN) ||
818                     (Parity == FACE_IOSS_Serial_PARITY_MARK) ||
819                     (Parity == FACE_IOSS_Serial_PARITY_SPACE)) {
820                     parameters.items[i].value.discriminator = FACE::IOSS::
    Serial::FACE_USHORT;
821                     parameters.items[i].value.values.ushort_value = Parity;
822                 } else {
823                     return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
824                     return;
825                 }
826                 break;
827             }
828             default: {
829                 // Unrecognized parameter ID
830                 return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
831                 return;
832             }
833         }
834     }
835     return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
836 }
```

```
837
838
839  /*
840   *
841   *  GETTERS AND SETTERS
842   *
843   */
844
845 FACE_Serial::~FACE_Serial() {
846
847 }
848
849 long FACE_Serial::getBaudRate() const {
850     return baudRate;
851 }
852
853 short FACE_Serial::getDataBits() const {
854     return dataBits;
855 }
856
857 short FACE_Serial::getStopBits() const {
858     return stopBits;
859 }
860
861 short FACE_Serial::getParity() const {
862     return parity;
863 }
864
865 short FACE_Serial::getFlowControl() const {
866     return flowControl;
867 }
868
869 bool FACE_Serial::isEcho() const {
870     return echo;
871 }
872
873 bool FACE_Serial::isTranslateLf() const {
874     return translateLF;
875 }
876
877 bool FACE_Serial::isTranslateEchoCr() const {
878     return translateEchoCR;
879 }
880
881 const char *FACE_Serial::getDeviceName() const {
882     return deviceName;
883 }
884
885
886 /*
887 void IO_Service_FACE_Serial::Get_Connection_Status(
888         FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
889         FACE::IOSS::Serial::CONNECTION_STATUS_TYPE& status,
890         FACE::RETURN_CODE_TYPE::Value& return_code)
891 {
892     if (isHandleUnknown(handle)) {
893         return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
894         return;
895     }
896
897
898     status = QueryConnectionStatus(handle);
899     return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
900 }
901
902 void IO_Service_FACE_Serial::Get_Bus_Status(
903         FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
904         FACE::IOSS::Serial::BUS_STATUS_TYPE& bus_status_value,
905         FACE::RETURN_CODE_TYPE::Value& return_code)
906 {
907     if (isHandleUnknown(handle)) {
908         return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
909         return;
910     }
911     bus_status_value = QueryBusStatus(handle);
```

```
912     return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
913 }
914
915 */
916
917 //void IO_Service_FACE_Serial::Register_Notification_Event(
918 void FACE::IO_Service_FACE_Serial::Register_Notification_Event(
919         FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
920         FACE::IO_Callback& io_callback,
921         FACE::RETURN_CODE_TYPE& return_code)
922 {
923     if (isHandleUnknown(handle)) {
924         return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
925         return;
926     }
927
928
929     int r = AssociateCallbackToHandle(handle,io_callback);
930     if(r==0){
931         return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
932     }
933     else if(r==2){
934         return_code = FACE::NO_ACTION;
935     }
936
937 }
938 //void IO_Service_FACE_Serial::Unregister_Notification_Event(
939 void FACE::IO_Service_FACE_Serial::Unregister_Notification_Event(
940         FACE::IOSS::CONNECTION_HANDLE_TYPE handle,
941         FACE::RETURN_CODE_TYPE& return_code)
942 {
943     if (isHandleUnknown(handle)) {
944         return_code = FACE::RETURN_CODE_TYPE::INVALID_PARAM;
945         return;
946     }
947     int r = DisassociateCallbackFromHandle(handle);
948
949     if(r==0){
950         return_code = FACE::RETURN_CODE_TYPE::NO_ERROR;
951     }
952     else if(r==2){
953         return_code = FACE::NO_ACTION;
954     }
955 }
```

A.5 FACE-External Sensors Simulator

A.5.1 AC_SIM

A.5.1.1 arrowlabel.h

```
1  #ifndef ARROWLABEL_H
2  #define ARROWLABEL_H
3
4  #include <QApplication>
5  #include <QLineEdit>
6  #include <QVBoxLayout>
7  #include <QWidget>
8  #include <QKeyEvent>
9  #include <QIntValidator>
10
11 class ArrowLabel: public QLineEdit {
12     Q_OBJECT
13
14 public:
15     explicit ArrowLabel(int inc, bool isUpDown, bool isLeftRight, int sat[4],
       QWidget *parent = nullptr);
16     void setValue(int value, char opz);
```

```
17
18 protected:
19     void keyPressEvent(QKeyEvent *event) override;
20     //void paintEvent(QPaintEvent *event) override;
21
22 private:
23     int valueUD;
24     int valueRL;
25     int inc;
26     bool isUpDown;
27     bool isLeftRight;
28     int sat[4];
29
30 signals:
31     void buttonClicked();
32 };
33
34 #endif
```

A.5.1.2 arrowlabel.cpp

```
1  #include "arrowlabel.h"
2
3
4  ArrowLabel::ArrowLabel(int inc, bool isUpDown, bool isLeftRight, int sat[4],
       QWidget *parent) :
5      QLineEdit(parent)
6      , valueUD(0)
7      , valueRL(0)
8      , inc(inc)
9      , isUpDown(isUpDown)
10     , isLeftRight(isLeftRight)
11 {
12     this->setValidator(new QIntValidator(this));
13
14     this->setFixedWidth(50);
15
16     if(this->isUpDown && !this->isLeftRight){
17         this->setText(QString::number(valueUD));
18     }
19     else if(!this->isUpDown && this->isLeftRight){
20         this->setText(QString::number(valueRL)+'/'+QString::number(-valueRL));
21     }
22     else{
23         this->setText(QString::number(valueUD)+'/'+QString::number(valueRL));
24     }
25
26     for(short i=0; i<4; i++){
27         this->sat[i] = sat[i];
28     }
29 }
30
31
32 void ArrowLabel::keyPressEvent(QKeyEvent *event){
33
34     bool valueChangedFlag = false;
35
36     if(this->isUpDown && !this->isLeftRight){ //throttle
37         if (event->key() == Qt::Key_Up) {
38             valueUD+=inc;
39             if(valueUD>sat[1]){
40                 valueUD=sat[1];
41             }
42             valueChangedFlag = true;
43         } else if (event->key() == Qt::Key_Down) {
44             valueUD-=inc;
45             if(valueUD<sat[0]){
46                 valueUD=sat[0];
47             }
48             valueChangedFlag = true;
49         }
```

```
50      }
51      else if(this->isLeftRight && !this->isUpDown){ //pedal
52          if (event->key() == Qt::Key_Left) {
53              valueRL-=inc;
54              if(valueRL<sat[2]){
55                  valueRL=sat[2];
56              }
57              valueChangedFlag = true;
58          } else if (event->key() == Qt::Key_Right) {
59              valueRL+=inc;
60              if(valueRL>sat[3]){
61                  valueRL=sat[3];
62              }
63              valueChangedFlag = true;
64          }
65      }
66      else{ //yoke
67          if (event->key() == Qt::Key_Up) {
68              valueUD+=inc;
69              if(valueUD>sat[1]){
70                  valueUD=sat[1];
71              }
72              valueChangedFlag = true;
73          } else if (event->key() == Qt::Key_Down) {
74              valueUD-=inc;
75              if(valueUD<sat[0]){
76                  valueUD=sat[0];
77              }
78              valueChangedFlag = true;
79          }
80          else if (event->key() == Qt::Key_Left) {
81              valueRL-=inc;
82              if(valueRL<sat[2]){
83                  valueRL=sat[2];
84              }
85              valueChangedFlag = true;
86          } else if (event->key() == Qt::Key_Right) {
87              valueRL+=inc;
88              if(valueRL>sat[3]){
89                  valueRL=sat[3];
90              }
91              valueChangedFlag = true;
92          }
93      }
94
95
96      if (valueChangedFlag) {
97          if(this->isUpDown && !this->isLeftRight){ //throttle
98              this->setText(QString::number(valueUD));
99          }
100         else if(this->isLeftRight && !this->isUpDown){ //pedal
101             this->setText(QString::number(valueRL)+'/'+QString::number(-
    valueRL));
102         }
103         else{ //yoke
104             this->setText(QString::number(valueUD)+'/'+QString::number(valueRL
    ));
105         }
106
107         emit buttonClicked();
108     }
109 }
110
111 void ArrowLabel::setValue(int value, char opz){
112     switch(opz){
113     case 0:
114         this->valueUD = value;
115         break;
116     case 1:
117         this->valueRL = value;
118         break;
119     }
120 }
```

A.5.1.3 inputscreen.h

```
#ifndef INPUTSCREEN_H
#define INPUTSCREEN_H

#include <QWidget>
#include <QLabel>
#include <QPushButton>
#include <QLineEdit>
#include <QGridLayout>
#include <QIntValidator>
#include <QDoubleValidator>

#define MAX_ALTITUDE 30000.0f
//#define MIN_ALTITUDE -30000.0f
#define MAX_LON 180.0
#define MAX_LAT 90.0
//#define MIN_LON -180.0;
//#define MIN_LAT -90.0;

class InputScreen : public QWidget
{
    Q_OBJECT
public:
    explicit InputScreen(QWidget *parent = nullptr);
    ~InputScreen();


private:

    QLabel *lat_label;
    QLabel *lon_label;
    QLabel *alt_label;

    QLineEdit *lat_text_deg;
    QLineEdit *lat_text_min;
    QLineEdit *lat_text_sec;

    QLineEdit *lon_text_deg;
    QLineEdit *lon_text_min;
    QLineEdit *lon_text_sec;

    QLineEdit *alt_text;

    QPushButton *set_btn;

public:
    QPushButton *get_button();
    QLineEdit *get_lat_deg();
    QLineEdit *get_lat_min();
    QLineEdit *get_lat_sec();
    QLineEdit *get_lon_deg();
    QLineEdit *get_lon_min();
    QLineEdit *get_lon_sec();
    QLineEdit *get_alt();

signals:
    void set_new_coordinates();

};

#endif // INPUTSCREEN_H
```

A.5.1.4 inputscreen.cpp

```
#include "inputscreen.h"

InputScreen::InputScreen(QWidget *parent)
    : QWidget{parent}
{
```

```
6       lat_label = new QLabel("Latitude", this);
7       lon_label = new QLabel("Longitude", this);
8       alt_label = new QLabel("Altitude", this);
9
10      QLabel *lat_label_s1 = new QLabel("°", this);
11      QLabel *lat_label_s2 = new QLabel("'", this);
12      QLabel *lat_label_s3 = new QLabel("''", this);
13      QLabel *lon_label_s1 = new QLabel("°", this);
14      QLabel *lon_label_s2 = new QLabel("'", this);
15      QLabel *lon_label_s3 = new QLabel("''", this);
16      QLabel *alt_label_s = new QLabel("feet", this);
17
18      lat_text_deg = new QLineEdit("", this);
19      lat_text_min = new QLineEdit("", this);
20      lat_text_sec = new QLineEdit("", this);
21      lon_text_deg = new QLineEdit("", this);
22      lon_text_min = new QLineEdit("", this);
23      lon_text_sec = new QLineEdit("", this);
24      alt_text = new QLineEdit("", this);
25
26      lat_text_deg->setFixedWidth(10);
27      lat_text_min->setFixedWidth(10);
28      lat_text_sec->setFixedWidth(10);
29      lon_text_deg->setFixedWidth(10);
30      lon_text_min->setFixedWidth(10);
31      lon_text_sec->setFixedWidth(10);
32      alt_text->setFixedWidth(35);
33
34
35      QValidator *val_deg_lat = new QIntValidator(-MAX_LAT, MAX_LAT, this);
36      QValidator *val_deg_lon = new QIntValidator(-MAX_LON, MAX_LON, this);
37      QValidator *val_min = new QIntValidator(0, 60, this);
38      QValidator *val_sec = new QIntValidator(0, 60, this);
39      lat_text_deg->setValidator(val_deg_lat);
40      lon_text_deg->setValidator(val_deg_lon);
41      lat_text_min->setValidator(val_min);
42      lon_text_min->setValidator(val_min);
43      lat_text_sec->setValidator(val_sec);
44      lon_text_sec->setValidator(val_sec);
45
46      QValidator *val_alt = new QIntValidator(0, MAX_ALTITUDE, this);
47      alt_text->setValidator(val_alt);
48
49      lat_text_deg->setFixedWidth(80);
50      lon_text_deg->setFixedWidth(80);
51      lat_text_min->setFixedWidth(80);
52      lon_text_min->setFixedWidth(80);
53      lat_text_sec->setFixedWidth(80);
54      lon_text_sec->setFixedWidth(80);
55      alt_text->setFixedWidth(80);
56
57      set_btn = new QPushButton("SET", this);
58      set_btn->setStyleSheet(
59          "background-color: lightblue; "
60          "color: blue; "
61          "font-weight: bold; "
62          );
63
64      QGridLayout *l = new QGridLayout(this);
65
66
67
68      l->addWidget(lat_label, 0, 0, 1, 1);
69      l->addWidget(lat_text_deg, 0, 1, 1, 1);
70      l->addWidget(lat_label_s1, 0, 2, 1, 1);
71      l->addWidget(lat_text_min, 0, 3, 1, 1);
72      l->addWidget(lat_label_s2, 0, 4, 1, 1);
73      l->addWidget(lat_text_sec, 0, 5, 1, 1);
74      l->addWidget(lat_label_s3, 0, 6, 1, 1);
75
76      l->addWidget(lon_label, 1, 0, 1, 1);
77      l->addWidget(lon_text_deg, 1, 1, 1, 1);
78      l->addWidget(lon_label_s1, 1, 2, 1, 1);
79      l->addWidget(lon_text_min, 1, 3, 1, 1);
80      l->addWidget(lon_label_s2, 1, 4, 1, 1);
```

```
81      l->addWidget(lon_text_sec, 1, 5, 1, 1);
82      l->addWidget(lon_label_s3, 1, 6, 1, 1);
83
84      l->addWidget(alt_label, 2, 0, 1, 1);
85      l->addWidget(alt_text, 2, 1, 1, 1);
86      l->addWidget(alt_label_s, 2, 2, 1, 1);
87
88      l->addWidget(set_btn, 0, 7, 3, 2);
89
90  }
91
92  InputScreen::~InputScreen(){}
93
94  QPushButton *InputScreen::get_button(){
95      return this->set_btn;
96  }
97
98  // GETTERS
99  QLineEdit *InputScreen::get_lat_deg(){
100     return this->lat_text_deg;
101 }
102 QLineEdit *InputScreen::get_lat_min(){
103     return this->lat_text_min;
104 }
105 QLineEdit *InputScreen::get_lat_sec(){
106     return this->lat_text_sec;
107 }
108 QLineEdit *InputScreen::get_lon_deg(){
109     return this->lon_text_deg;
110 }
111 QLineEdit *InputScreen::get_lon_min(){
112     return this->lon_text_min;
113 }
114 QLineEdit *InputScreen::get_lon_sec(){
115     return this->lon_text_sec;
116 }
117 QLineEdit *InputScreen::get_alt(){
118     return this->alt_text;
119 }
```

A.5.1.5 simscreen_egi.h

```
1   #ifndef SIMSCREEN_EGI_H
2   #define SIMSCREEN_EGI_H
3
4   #include <QWidget>
5   #include <QLabel>
6   #include <QPushButton>
7
8   #include "arrowlabel.h"
9   #include "fcntl.h"
10  #include "FACE_LAYERS/IOSS/Serial/include/Serial.h"
11  #include "FACE_LAYERS/PSSS/sim_utils.h"
12  #include "sim_constant.h"
13  #include "FACE_LAYERS/PSSS/DPM/backup.h"
14
15
16  typedef struct kynematics_s{
17      speed_t speed_NED;
18      acc_t acc_NED;
19      double speed;
20  }kynematics_t;
21
22  class SimScreen_EGI : public QWidget
23  {
24      Q_OBJECT
25  public:
26      explicit SimScreen_EGI(QWidget *parent = nullptr);
27      ~SimScreen_EGI();
28
29
```

```
30      void set_heading(float num);
31      void set_pitch(short p);
32      void set_roll(short r);
33      void set_yaw(short y);
34
35      void set_speed_NED(speed_t s);
36      void set_speed(double s);
37
38      float get_heading();
39      short get_pitch();
40      short get_roll();
41      short get_yaw();
42
43      double get_lat();
44      double get_lon();
45      int get_alt();
46
47      void set_lat(double lat);
48      void set_lon(double lon);
49      void set_alt(int alt);
50
51      speed_t get_speed_NED();
52      double get_speed();
53
54      void set_all_data(kynematics_t kyn, float heading, short pitch, short roll
        , short yaw, double lat, double lon, int alt);
55      void set_all_data_NO_POS(kynematics_t kyn, float heading, short pitch,
        short roll, short yaw);
56
57      void send_data();
58
59      bool switch_status();
60      bool get_status();
61
62  private:
63
64      QString double2dms_string(const double value);
65      void measure();
66
67      QGridLayout *layoutTop;
68      QGridLayout *layoutBottom;
69      QVBoxLayout *layout;
70
71      QLabel *pitch_l;
72      QLabel *roll_l;
73      QLabel *yaw_l;
74      QLabel *hd_l;
75
76      QLineEdit *pitch_tx;
77      QLineEdit *roll_tx;
78      QLineEdit *yaw_tx;
79      QLineEdit *hd_text;
80
81      QLabel *speed_x_l;
82      QLabel *speed_y_l;
83      QLabel *speed_z_l;
84
85      QLineEdit *speed_x;
86      QLineEdit *speed_y;
87      QLineEdit *speed_z;
88
89      QLabel *acc_x_l;
90      QLabel *acc_y_l;
91      QLabel *acc_z_l;
92
93      QLineEdit *acc_x;
94      QLineEdit *acc_y;
95      QLineEdit *acc_z;
96
97      QLabel *lat_label;
98      QLabel *lon_label;
99      QLabel *alt_label;
100
101     QLineEdit *lat_tx;
102     QLineEdit *lon_tx;
```

```
103     QLineEdit *alt_tx;
104
105     QLabel *speed_label;
106     QLineEdit *speed_text;
107
108
109     float heading; //radians
110
111     short pitch;
112     short roll;
113     short yaw;
114
115     double lat0;
116     double lon0;
117     int alt0;
118
119
120     double delta_lat;
121     double delta_lon;
122     float delta_alt;
123
124     double speed;
125     speed_t speed_NED;
126     pos_t pos_NED;
127     acc_t acc_NED;
128
129     bool status;
130
131 #ifdef SERIAL
132     //SERIAL
133     FACE_Serial serial;
134     FACE::IOSS::Serial::WRITE_PAYLOAD_TYPE wr;
135     FACE::RETURN_CODE_TYPE return_code;
136
137     FACE::IOSS::Serial::CONNECTION_NAME_TYPE name;
138     FACE::TIMEOUT_TYPE timeout = FACE::IOSS::Serial::WAIT_FOREVER;
139     FACE::IOSS::CONNECTION_HANDLE_TYPE handle;
140
141 #ifdef COMPRESS
142     char *packet;
143 #endif
144
145 #endif
146 };
147
148
149
150 #endif // SIMSCREEN_EGI_H
```

A.5.1.6 simscreen_egi.cpp

```
1  #include "simscreen_egi.h"
2  #include "FACE_LAYERS/PSSS/sim_utils.h"
3
4  SimScreen_EGI::SimScreen_EGI(QWidget *parent)
5      : QWidget{parent}
6  {
7
8      pitch_l = new QLabel("Pitch", this);
9      roll_l = new QLabel("Roll", this);
10     yaw_l = new QLabel("Yaw", this);
11     hd_l = new QLabel("Heading", this);
12
13     pitch_tx = new QLineEdit("", this);
14     roll_tx = new QLineEdit("", this);
15     yaw_tx = new QLineEdit("", this);
16     hd_text = new QLineEdit("", this);
17
18     speed_x_l = new QLabel("Speed X", this);
19     speed_y_l = new QLabel("Speed Y", this);
20     speed_z_l = new QLabel("Speed Z", this);
```

```
21
22      speed_x = new QLineEdit("", this);
23      speed_y = new QLineEdit("", this);
24      speed_z = new QLineEdit("", this);
25
26      acc_x_l = new QLabel("Accel X", this);
27      acc_y_l = new QLabel("Accel Y", this);
28      acc_z_l = new QLabel("Accel Z", this);
29
30      acc_x = new QLineEdit("", this);
31      acc_y = new QLineEdit("", this);
32      acc_z = new QLineEdit("", this);
33
34      lat_label = new QLabel("LAT", this);
35      lon_label = new QLabel("LON", this);
36      alt_label = new QLabel("ALT", this);
37
38      lat_tx = new QLineEdit("", this);
39      lon_tx = new QLineEdit("", this);
40      alt_tx = new QLineEdit("", this);
41
42      speed_label = new QLabel("Speed", this);
43      speed_text = new QLineEdit("", this);
44
45
46      QLabel *angle_label_s1 = new QLabel("°", this);
47      QLabel *angle_label_s2 = new QLabel("°", this);
48      QLabel *angle_label_s3 = new QLabel("°", this);
49      QLabel *speed_label_s = new QLabel("m/s", this);
50      QLabel *speed_label_s1 = new QLabel("feet/s", this);
51      QLabel *speed_label_s2 = new QLabel("feet/s", this);
52      QLabel *speed_label_s3 = new QLabel("feet/s", this);
53      QLabel *acc_label_s1 = new QLabel("feet/s^2", this);
54      QLabel *acc_label_s2 = new QLabel("feet/s^2", this);
55      QLabel *acc_label_s3 = new QLabel("feet/s^2", this);
56
57      QLabel *hd_label_s = new QLabel("°", this);
58
59      QLabel *lat_l_s = new QLabel("°", this);
60      QLabel *lon_l_s = new QLabel("°", this);
61      QLabel *alt_l_s = new QLabel("feet", this);
62
63
64      pitch_tx->setReadOnly(true);
65      roll_tx->setReadOnly(true);
66      yaw_tx->setReadOnly(true);
67      hd_text->setReadOnly(true);
68
69      speed_x->setReadOnly(true);
70      speed_y->setReadOnly(true);
71      speed_z->setReadOnly(true);
72
73      acc_x->setReadOnly(true);
74      acc_y->setReadOnly(true);
75      acc_z->setReadOnly(true);
76
77      lat_tx->setReadOnly(true);
78      lon_tx->setReadOnly(true);
79      alt_tx->setReadOnly(true);
80
81      speed_text->setReadOnly(true);
82
83      pitch_tx->setFixedWidth(70);
84      roll_tx->setFixedWidth(70);
85      yaw_tx->setFixedWidth(70);
86
87
88      speed_x->setFixedWidth(70);
89      speed_y->setFixedWidth(70);
90      speed_z->setFixedWidth(70);
91
92      acc_x->setFixedWidth(70);
93      acc_y->setFixedWidth(70);
94      acc_z->setFixedWidth(70);
95
```

```
96      lat_tx->setFixedWidth(80);
97      lon_tx->setFixedWidth(80);
98      alt_tx->setFixedWidth(60);
99
100     hd_text->setFixedWidth(90);
101     speed_text->setFixedWidth(100);
102
103     QWidget *layoutTop_w = new QWidget();
104     QWidget *layoutBottom_w = new QWidget();
105
106     layoutTop = new QGridLayout(layoutTop_w);
107     layoutBottom = new QGridLayout(layoutBottom_w);
108
109
110     // ANGLES
111     layoutTop->addWidget(pitch_l, 0, 0, 1, 1);
112     layoutTop->addWidget(pitch_tx, 0, 1, 1, 1);
113     layoutTop->addWidget(angle_label_s1, 0, 2, 1, 1);
114
115     layoutTop->addWidget(roll_l, 1, 0, 1, 1);
116     layoutTop->addWidget(roll_tx, 1, 1, 1, 1);
117     layoutTop->addWidget(angle_label_s2, 1, 2, 1, 1);
118
119     layoutTop->addWidget(yaw_l, 2, 0, 1, 1);
120     layoutTop->addWidget(yaw_tx, 2, 1, 1, 1);
121     layoutTop->addWidget(angle_label_s3, 2, 2, 1, 1);
122
123
124     // COORDS
125     layoutTop->addWidget(lat_label, 0, 3, 1, 1);
126     layoutTop->addWidget(lat_tx, 0, 4, 1, 1);
127     layoutTop->addWidget(lat_l_s, 0, 5, 1, 1);
128
129     layoutTop->addWidget(lon_label, 1, 3, 1, 1);
130     layoutTop->addWidget(lon_tx, 1, 4, 1, 1);
131     layoutTop->addWidget(lon_l_s, 1, 5, 1, 1);
132
133     layoutTop->addWidget(alt_label, 2, 3, 1, 1);
134     layoutTop->addWidget(alt_tx, 2, 4, 1, 1);
135     layoutTop->addWidget(alt_l_s, 2, 5, 1, 1);
136
137
138     //SPEED
139     layoutTop->addWidget(speed_x_l, 3, 0, 1, 1);
140     layoutTop->addWidget(speed_x, 3, 1, 1, 1);
141     layoutTop->addWidget(speed_label_s1, 3, 2, 1, 1);
142
143     layoutTop->addWidget(speed_y_l, 4, 0, 1, 1);
144     layoutTop->addWidget(speed_y, 4, 1, 1, 1);
145     layoutTop->addWidget(speed_label_s2, 4, 2, 1, 1);
146
147     layoutTop->addWidget(speed_z_l, 5, 0, 1, 1);
148     layoutTop->addWidget(speed_z, 5, 1, 1, 1);
149     layoutTop->addWidget(speed_label_s3, 5, 2, 1, 1);
150
151     //ACC
152     layoutTop->addWidget(acc_x_l, 3, 3, 1, 1);
153     layoutTop->addWidget(acc_x, 3, 4, 1, 1);
154     layoutTop->addWidget(acc_label_s1, 3, 5, 1, 1);
155
156     layoutTop->addWidget(acc_y_l, 4, 3, 1, 1);
157     layoutTop->addWidget(acc_y, 4, 4, 1, 1);
158     layoutTop->addWidget(acc_label_s2, 4, 5, 1, 1);
159
160     layoutTop->addWidget(acc_z_l, 5, 3, 1, 1);
161     layoutTop->addWidget(acc_z, 5, 4, 1, 1);
162     layoutTop->addWidget(acc_label_s3, 5, 5, 1, 1);
163
164     //AC_SPEED
165     layoutBottom->addWidget(speed_label, 0, 0, 1, 1);
166     layoutBottom->addWidget(speed_text, 0, 1, 1, 1);
167     layoutBottom->addWidget(speed_label_s, 0, 2, 1, 1);
168
169     layoutBottom->addWidget(hd_l, 1, 0, 1, 1);
170     layoutBottom->addWidget(hd_text, 1, 1, 1, 1);
```

```
171     layoutBottom->addWidget(hd_label_s, 1, 2, 1, 1);
172 
173     //HEADING
174 
175     status = true;
176 
177 #ifdef SERIAL
178     //SERIAL
179 
180     FACE::CONFIGURATION_RESOURCE config = "AC_SIM/serial_EGI";
181     serial.Initialize(config, return_code);
182     if(return_code!=FACE::NO_ERROR){
183         perror("No Initialization possible!!\n");
184     }
185 
186     wr.payload = (FACE::IOSS::Serial::PAYLOAD_DATA_MSG_TYPE) malloc(sizeof(
    char)*EGI_PACKET_SIZE);
187     memset((void*) (wr.payload), 0, EGI_PACKET_SIZE);
188     serial.Open_Connection(name, timeout, handle, return_code);
189 
190     if(return_code!=FACE::NO_ERROR){
191         perror("No Opening possible!!\n");
192     }
193 
194 #ifdef COMPRESS
195     packet = (char*) malloc((size_t)EGI_PACKET_SIZE*sizeof(char));
196 #endif
197 
198 #endif
199 
200     QWidget *w = new QWidget(this);
201     layout = new QVBoxLayout(w);
202     layout->addWidget(layoutTop_w);
203     layout->addWidget(layoutBottom_w);
204 
205     setLayout(layout);
206 
207 }
208 
209 
210 void SimScreen_EGI::set_heading(float hd){
211     this->heading = hd;
212     this->hd_text->setText(QString::number((int)rad2deg(this->heading)));
213 }
214 
215 void SimScreen_EGI::set_pitch(short p){
216     this->pitch = p;
217     this->pitch_tx->setText(QString::number(this->pitch));
218 }
219 
220 void SimScreen_EGI::set_roll(short r){
221     this->roll = r;
222     this->roll_tx->setText(QString::number(this->roll));
223 }
224 
225 void SimScreen_EGI::set_yaw(short y){
226     this->yaw = y;
227     this->yaw_tx->setText(QString::number(this->yaw));
228 }
229 
230 void SimScreen_EGI::set_speed_NED(speed_t s){
231 
232     speed_t old_speed = this->speed_NED;
233 
234     this->speed_NED = s;
235     if(abs(this->speed_NED.X)<0.1)
236         this->speed_x->setText(QString::number(0));
237     else
238         this->speed_x->setText(QString::number(this->speed_NED.X));
239 
240     if(abs(this->speed_NED.Y)<0.1)
241         this->speed_y->setText(QString::number(0));
242     else
243         this->speed_y->setText(QString::number(this->speed_NED.Y));
244 
```

```
245     if(abs(this->speed_NED.Z)<0.1)
246         this->speed_z->setText(QString::number(0));
247     else
248         this->speed_z->setText(QString::number(this->speed_NED.Z));
249
250
251     acc_NED.X = (abs(speed_NED.X) - abs(old_speed.X))/delta_T_sec;
252     acc_NED.Y = (abs(speed_NED.Y) - abs(old_speed.Y))/delta_T_sec;
253     acc_NED.Z = (abs(speed_NED.Z) - abs(old_speed.Z))/delta_T_sec;
254
255
256     this->acc_x->setText(QString::number(((std::abs(acc_NED.X) > 0.01) ?
        acc_NED.X : 0.0), 'f', 2));
257     this->acc_y->setText(QString::number(((std::abs(acc_NED.Y) > 0.01) ?
        acc_NED.Y : 0.0), 'f', 2));
258     this->acc_z->setText(QString::number(((std::abs(acc_NED.Z) > 0.01) ?
        acc_NED.Z : 0.0), 'f', 2));
259
260 }
261
262 void SimScreen_EGI::set_speed(double s){
263     this->speed = s;
264     this->speed_text->setText(QString::number((int)s));
265 }
266
267 float SimScreen_EGI::get_heading(){
268     return this->heading;
269 }
270 short SimScreen_EGI::get_pitch(){
271     return this->pitch;
272 }
273 short SimScreen_EGI::get_roll(){
274     return this->roll;
275 }
276 short SimScreen_EGI::get_yaw(){
277     return this->yaw;
278 }
279
280 double SimScreen_EGI::get_lat(){
281     return (this->delta_lat+this->lat0);
282 }
283
284 double SimScreen_EGI::get_lon(){
285     return (this->delta_lon+this->lon0);
286 }
287
288 int SimScreen_EGI::get_alt(){
289     return (this->delta_alt+this->alt0);
290 }
291
292 speed_t SimScreen_EGI::get_speed_NED(){
293     return this->speed_NED;
294 }
295
296 void SimScreen_EGI::set_lat(double lat){
297     this->lat0 = lat;
298     this->delta_lat = 0;
299     this->lat_tx->setText(double2dms_string(this->lat0));
300     //this->lat_tx->setText(QString::number(this->lat0));
301
302 }
303
304 void SimScreen_EGI::set_lon(double lon){
305     this->lon0 = lon;
306     this->delta_lon = 0;
307     this->lon_tx->setText(double2dms_string(this->lon0));
308     //this->lon_tx->setText(QString::number(this->lon0));
309 }
310
311 void SimScreen_EGI::set_alt(int alt){
312     this->alt0 = alt;
313     this->delta_alt = 0;
314     this->alt_tx->setText(QString::number(this->alt0));
315 }
316
```

```
317
318 void SimScreen_EGI::set_all_data(kynematics_t kyn, float heading, short p,
    short r,
319                                  short y, double lat, double lon, int alt){
320
321     set_heading(heading);
322     set_pitch(p);
323     set_roll(r);
324     set_yaw(y);
325     set_speed_NED(kyn.speed_NED);
326     set_speed(kyn.speed);
327     set_lat(lat);
328     set_lon(lon);
329     set_alt(alt);
330 }
331
332 void SimScreen_EGI::set_all_data_NO_POS(kynematics_t kyn, float heading, short
     p, short r, short y){
333
334     set_heading(heading);
335     set_pitch(p);
336     set_roll(r);
337     set_yaw(y);
338     set_speed_NED(kyn.speed_NED);
339     set_speed(kyn.speed);
340     measure();
341
342     //printf("EGI: %f %f\n", this->lat0+this->delta_lat, this->lon0+this->
    delta_lon);
343 }
344
345 void SimScreen_EGI::send_data(){
346
347     if(status){
348
349         std::string c1_bin = numericToBinary<float>((float)((delta_lon+lon0) *
     (1.0f / (1 << 31))), 32);
350         std::string c2_bin = numericToBinary<float>((float)((delta_lat+lat0) *
     (1.0f / (1 << 31))), 32);
351         std::string alt_bin = numericToBinary<float>(((float)(delta_alt+alt0)
    * (1.0f / (1 << 3))), 20);
352         std::string s_bin = numericToBinary<float>((int)speed * (1.0f / (1 <<
    0)), 16);
353         std::string hd_bin = numericToBinary<float>(heading * (1.0f / (1 <<
    15)), 16);
354         std::string p_bin = numericToBinary<float>((float)pitch * (1.0f / (1
    << 15)), 16);
355         std::string r_bin = numericToBinary<float>((float)roll * (1.0f / (1 <<
     15)), 16);
356         std::string y_bin = numericToBinary<float>((float)yaw * (1.0f / (1 <<
    15)), 16);
357         std::string s_x_bin = numericToBinary<float>(speed_NED.X * (1.0f / (1
    << 18)), 32);
358         std::string s_y_bin = numericToBinary<float>(speed_NED.Y * (1.0f / (1
    << 18)), 32);
359         std::string s_z_bin = numericToBinary<float>(speed_NED.Z * (1.0f / (1
    << 18)), 32);
360         std::string a_x_bin = numericToBinary<float>(acc_NED.X * (1.0f / (1 <<
     5)), 16);
361         std::string a_y_bin = numericToBinary<float>(acc_NED.Y * (1.0f / (1 <<
     5)), 16);
362         std::string a_z_bin = numericToBinary<float>(acc_NED.Z * (1.0f / (1 <<
     5)), 16);
363 #ifdef SERIAL
364
365 #ifdef COMPRESS
366
367         char *msg = (char*) calloc((size_t)EGI_PACKET_SIZE*8, sizeof(char));
368
369         strcat(msg, c1_bin.c_str());
370         strcat(msg, c2_bin.c_str());
371         strcat(msg, alt_bin.c_str());
372         strcat(msg, s_bin.c_str());
373         strcat(msg, hd_bin.c_str());
374         strcat(msg, p_bin.c_str());
```

```
375         strcat(msg, r_bin.c_str());
376         strcat(msg, y_bin.c_str());
377         strcat(msg, s_x_bin.c_str());
378         strcat(msg, s_y_bin.c_str());
379         strcat(msg, s_z_bin.c_str());
380         strcat(msg, a_x_bin.c_str());
381         strcat(msg, a_y_bin.c_str());
382         strcat(msg, a_z_bin.c_str());
383 
384         memset((void*) packet, 0, EGI_PACKET_SIZE);
385         compress(packet, msg, EGI_PACKET_SIZE);
386         memcpy(wr.payload, packet, EGI_PACKET_SIZE);
387 
388         printf("SC_EGI => ");
389         for(int i=0; i<EGI_PACKET_SIZE; i++){
390             printf("%d ", (char) packet[i]);
391         }
392         printf("\n");
393         fflush(stdout);
394 
395         this->serial.Write(handle, FACE::IOSS::Serial::WAIT_FOREVER, &wr,
    EGI_PACKET_SIZE, return_code);
396         free(msg);
397 #else
398         memcpy(wr.payload, (c1_bin+c2_bin+alt_bin+s_bin+hd_bin+p_bin+r_bin+
    y_bin+
399                             s_x_bin+s_y_bin+s_z_bin+a_x_bin+a_y_bin+a_z_bin).
    c_str(), EGI_PACKET_SIZE);
400         this->serial.Write(handle, FACE::IOSS::Serial::WAIT_FOREVER, &wr,
    EGI_PACKET_SIZE, return_code);
401 #endif
402 
403 #endif
404 
405     }
406     else{
407 #ifdef SERIAL
408         this->serial.Flush(handle, return_code);
409 #endif
410     }
411 
412 }
413 
414 SimScreen_EGI::~SimScreen_EGI(){
415 #ifdef SERIAL
416     serial.Close_Connection(handle, return_code);
417 #ifdef COMPRESS
418     free(packet);
419 #endif
420 #endif
421 }
422 
423 void SimScreen_EGI::measure(){
424 
425     double lat_tot = this->lat0+this->delta_lat;
426     double lon_tot = this->lon0+this->delta_lon;
427 
428     /*nedToLatLon2(&(this->delta_lat), &(this->delta_lon),
429                  this->speed_NED.X*delta_T_sec,
430                  this->speed_NED.Y*delta_T_sec);*/
431 
432     nedToLatLon2(&lat_tot, &lon_tot,
433                  this->speed_NED.X*delta_T_sec,
434                  this->speed_NED.Y*delta_T_sec);
435 
436     this->delta_lat = lat_tot-this->lat0;
437     this->delta_lon = lon_tot-this->lon0;
438 
439     double noise = ((double) rand() / RAND_MAX) * 2 * noiseMagnitude_h -
    noiseMagnitude_h;
440 
441     this->delta_alt += (float) (this->speed_NED.Z*delta_T_sec*MT2FT)+noise
    /10.0;
442 
443     this->lat_tx->setText(double2dms_string(lat_tot));
```

```
444     this->lon_tx->setText(double2dms_string(lon_tot));
445
446     //this->lat_tx->setText(double2dms_string(this->lat0+this->delta_lat));
447     //this->lon_tx->setText(double2dms_string(this->lon0+this->delta_lon));
448
449     this->alt_tx->setText(QString::number(this->alt0+(int)this->delta_alt));
450 }
451
452 bool SimScreen_EGI::switch_status(){
453     status = !status;
454     return status;
455 }
456
457 bool SimScreen_EGI::get_status(){
458     return status;
459 }
460
461 QString SimScreen_EGI::double2dms_string(const double value){
462     Coordinate_t c = decToDMS_r(value);
463     QString ris =  QString("%1° %2' %3''")
464                         .arg(c.degree)
465                         .arg(c.min)
466                         .arg((int)c.sec);
467     return ris;
468 }
```

A.5.1.7 simscreen_gps.h

```
1  #ifndef SIMSCREEN_GPS_H
2  #define SIMSCREEN_GPS_H
3
4  #include <QWidget>
5  #include <QLabel>
6  #include <QPushButton>
7
8  #include "arrowlabel.h"
9  #include "fcntl.h"
10 #include "FACE_LAYERS/IOSS/Serial/include/Serial.h"
11 #include "FACE_LAYERS/PSSS/sim_utils.h"
12 #include "sim_constant.h"
13 #include "FACE_LAYERS/PSSS/DPM/backup.h"
14
15 class SimScreen_GPS : public QWidget
16 {
17     Q_OBJECT
18 public:
19     explicit SimScreen_GPS(QWidget *parent = nullptr);
20     ~SimScreen_GPS();
21
22     QLineEdit *get_lat_text();
23     QLineEdit *get_lon_text();
24
25     QLineEdit *get_alt_text();
26
27     void set_lat_text(Coordinate_t lat);
28     void set_lon_text(Coordinate_t lon);
29
30     void set_heading_text(float num);
31     void set_alt_text(int num);
32     void set_hd_text(float hd);
33
34     void set_all_data(Coordinate_t lat, Coordinate_t lon, int altitude,
35                       float heading, double speed);
36
37     void set_speed(double speed);
38     double get_speed();
39
40     void send_data();
41
42     bool switch_status();
43     bool get_status();
```

```
44
45 private:
46
47     QString dms_string(Coordinate_t c);
48
49     QGridLayout *layout;
50
51     QLabel *lat_label;
52     QLabel *lon_label;
53     QLabel *alt_label;
54     QLabel *speed_label;
55     QLabel *hd_label;
56
57     QLineEdit *lat_text;
58
59     QLineEdit *lon_text;
60
61
62     QLineEdit *alt_text;
63     QLineEdit *speed_text;
64     QLineEdit *hd_text;
65
66     Coordinate_t lon;
67     Coordinate_t lat;
68     int altitude;
69     float heading; //radians
70
71     double speed;
72     bool status;
73
74 #ifdef SERIAL
75     //SERIAL
76     FACE_Serial serial;
77     FACE::IOSS::Serial::WRITE_PAYLOAD_TYPE wr;
78     FACE::RETURN_CODE_TYPE return_code;
79
80     FACE::IOSS::Serial::CONNECTION_NAME_TYPE name;
81     FACE::TIMEOUT_TYPE timeout = FACE::IOSS::Serial::WAIT_FOREVER;
82     FACE::IOSS::CONNECTION_HANDLE_TYPE handle;
83
84 #ifdef COMPRESS
85     char *packet;
86 #endif
87
88 #endif
89 };
90
91
92
93 #endif // SIMSCREEN_GPS_H
```

A.5.1.8 simscreen_gps.cpp

```
1  #include "simscreen_gps.h"
2  #include "FACE_LAYERS/PSSS/sim_utils.h"
3
4  SimScreen_GPS::SimScreen_GPS(QWidget *parent)
5      : QWidget{parent}
6  {
7
8      lat_label = new QLabel("LAT", this);
9      lon_label = new QLabel("LON", this);
10     alt_label = new QLabel("ALT", this);
11     speed_label = new QLabel("Speed", this);
12     hd_label = new QLabel("Heading", this);
13
14     QLabel *lat_label_s1 = new QLabel("°", this);
15     QLabel *lon_label_s1 = new QLabel("°", this);
16     QLabel *alt_label_s = new QLabel("feet", this);
17     QLabel *speed_label_s = new QLabel("m/s", this);
18     QLabel *hd_label_s = new QLabel("°", this);
```

```
19
20
21      lat_text = new QLineEdit("", this);
22      lon_text = new QLineEdit("", this);
23      alt_text = new QLineEdit("", this);
24      speed_text = new QLineEdit("", this);
25      hd_text = new QLineEdit("", this);
26
27      lat_text->setReadOnly(true);
28      lon_text->setReadOnly(true);
29      alt_text->setReadOnly(true);
30      speed_text->setReadOnly(true);
31      hd_text->setReadOnly(true);
32
33      lat_text->setFixedWidth(80);
34      lon_text->setFixedWidth(80);
35      alt_text->setFixedWidth(70);
36      speed_text->setFixedWidth(90);
37      hd_text->setFixedWidth(90);
38
39      layout = new QGridLayout(this);
40
41      layout->addWidget(lat_label, 0, 0, 1, 1);
42      layout->addWidget(lat_text, 0, 1, 1, 1);
43      layout->addWidget(lat_label_s1, 0, 2, 1, 1);
44
45      layout->addWidget(lon_label, 1, 0, 1, 1);
46      layout->addWidget(lon_text, 1, 1, 1, 1);
47      layout->addWidget(lon_label_s1, 1, 2, 1, 1);
48
49      layout->addWidget(alt_label, 2, 0, 1, 1);
50      layout->addWidget(alt_text, 2, 1, 1, 1);
51      layout->addWidget(alt_label_s, 2, 2, 1, 1);
52
53      layout->addWidget(speed_label, 3, 0, 1, 1);
54      layout->addWidget(speed_text, 3, 1, 1, 1);
55      layout->addWidget(speed_label_s, 3, 2, 1, 1);
56
57      layout->addWidget(hd_label, 4, 0, 1, 1);
58      layout->addWidget(hd_text, 4, 1, 1, 1);
59      layout->addWidget(hd_label_s, 4, 2, 1, 1);
60
61      status = true;
62
63
64  #ifdef SERIAL
65      //SERIAL
66
67
68      FACE::CONFIGURATION_RESOURCE config = "AC_SIM/serial_GPS";
69      serial.Initialize(config, return_code);
70      if(return_code!=FACE::NO_ERROR){
71          perror("No Initialization possible!!\n");
72      }
73
74
75      wr.payload = (FACE::IOSS::Serial::PAYLOAD_DATA_MSG_TYPE) malloc(sizeof(
        char)*GPS_PACKET_SIZE);
76      memset((void*) (wr.payload), 0, GPS_PACKET_SIZE);
77      serial.Open_Connection(name, timeout, handle, return_code);
78
79      if(return_code!=FACE::NO_ERROR){
80          perror("No Opening possible!!\n");
81      }
82  #ifdef COMPRESS
83      packet = (char*) malloc((size_t) GPS_PACKET_SIZE*sizeof(char));
84  #endif
85
86  #endif
87
88      setLayout(layout);
89
90  }
91
92  QLineEdit *SimScreen_GPS::get_lat_text(){
```

```
93      return this->lat_text;
94  }
95
96  QLineEdit *SimScreen_GPS::get_lon_text(){
97      return this->lon_text;
98  }
99
100 QLineEdit *SimScreen_GPS::get_alt_text(){
101     return this->alt_text;
102 }
103
104 void SimScreen_GPS::set_lat_text(Coordinate_t lat){
105     this->lat.degree = lat.degree;
106     this->lat.min = lat.min;
107     this->lat.sec = lat.sec;
108     this->lat_text->setText(dms_string(this->lat));
109     //this->lat_text->setText(QString::number(dmsToDec_r(this->lat)));
110 }
111
112
113 void SimScreen_GPS::set_lon_text(Coordinate_t lon){
114     this->lon.degree = lon.degree;
115     this->lon.min = lon.min;
116     this->lon.sec = lon.sec;
117     this->lon_text->setText(dms_string(this->lon));
118     //this->lon_text->setText(QString::number(dmsToDec_r(this->lon)));
119 }
120
121
122 void SimScreen_GPS::set_alt_text(int num){
123     altitude = num;
124     this->alt_text->setText(QString::number(num));
125 }
126
127 void SimScreen_GPS::set_heading_text(float num){
128     this->heading = num;
129     this->hd_text->setText(QString::number((int)rad2deg(num)));
130 }
131
132 void SimScreen_GPS::set_speed(double speed){
133     this->speed = speed;
134     this->speed_text->setText(QString::number((int)speed));
135 }
136
137 double SimScreen_GPS::get_speed(){
138     return this->speed;
139 }
140
141 void SimScreen_GPS::set_all_data(Coordinate_t lat, Coordinate_t lon, int
    altitude,
142                                  float heading, double speed){
143     set_lat_text(lat);
144     set_lon_text(lon);
145     set_alt_text(altitude);
146     set_heading_text(heading);
147     set_speed(speed);
148 }
149
150 void SimScreen_GPS::send_data(){
151
152     if(status){
153         std::string c1_bin = numericToBinary<double>((dmsToDec_r(lon) * (1.0f
    / (1 << 31))), 32);
154         std::string c2_bin = numericToBinary<double>((dmsToDec_r(lat) * (1.0f
    / (1 << 31))), 32);
155         std::string alt_bin = numericToBinary<float>(((float)altitude * (1.0f
    / (1 << 3))), 20);
156         std::string s_bin = numericToBinary<float>((int)speed * (1.0f / (1 <<
    0)), 16);
157         std::string hd_bin = numericToBinary<float>(heading * (1.0f / (1 <<
    15)), 16);
158
159
160 #ifdef SERIAL
161
```

```
162 #ifdef COMPRESS
163         char *msg = (char*) calloc((size_t)GPS_PACKET_SIZE*8, sizeof(char));
164 
165         strcat(msg, c1_bin.c_str());
166         strcat(msg, c2_bin.c_str());
167         strcat(msg, alt_bin.c_str());
168         strcat(msg, s_bin.c_str());
169         strcat(msg, hd_bin.c_str());
170         strcat(msg, "000000000000"); //padding
171         //strcat(msg, "0000"); //padding
172 
173         printf("LEN %d\n", strlen(msg));
174 
175 
176         memset((void*) packet, 0, GPS_PACKET_SIZE);
177 
178         compress(packet, msg, GPS_PACKET_SIZE);
179         memcpy(wr.payload, packet, GPS_PACKET_SIZE);
180 
181         /*printf("SC_GPS => ");
182         for(int i=0; i<GPS_PACKET_SIZE; i++){
183             printf("%d ", (char) wr.payload[i]);
184         }
185         printf("\n");
186         fflush(stdout);*/
187 
188         this->serial.Write(handle, FACE::IOSS::Serial::WAIT_FOREVER, &wr,
    GPS_PACKET_SIZE, return_code);
189         free(msg);
190 #else
191         memcpy(wr.payload, (c1_bin+c2_bin+alt_bin+s_bin+hd_bin).c_str(),
    GPS_PACKET_SIZE);
192 
193         this->serial.Write(handle, FACE::IOSS::Serial::WAIT_FOREVER, &wr,
    GPS_PACKET_SIZE, return_code);
194 #endif
195 
196 #endif
197     }
198     else{
199 #ifdef SERIAL
200         this->serial.Flush(handle, return_code);
201 
202 #endif
203     }
204 }
205 
206 SimScreen_GPS::~SimScreen_GPS(){
207 #ifdef SERIAL
208     serial.Close_Connection(handle, return_code);
209 #ifdef COMPRESS
210     free(packet);
211 #endif
212 
213 #endif
214 }
215 
216 bool SimScreen_GPS::switch_status(){
217     status = !status;
218     return status;
219 }
220 
221 bool SimScreen_GPS::get_status(){
222     return status;
223 }
224 
225 QString SimScreen_GPS::dms_string(Coordinate_t c){
226     QString ris =  QString("%1° %2' %3''")
227                         .arg(c.degree)
228                         .arg(c.min)
229                         .arg((int)c.sec);
230     return ris;
231 }
```

A.5.1.9 mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTimer>
#include <QFormLayout>
#include <QFrame>
#include "simscreen_gps.h"
#include "simscreen_egi.h"
#include "inputscreen.h"
#include "FACE_LAYERS/PSSS/sim_utils.h"
//#include "FACE_LAYERS/PSSS/DPM/backup.h"
#include "math.h"
//#include "sim_constant.h"


QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

private slots:
    void handleArrowLabel();
    void handleSetButton();
    void simulate();
    void handle_EGI_switch();
    void handle_GPS_switch();

public slots:
#ifdef SERVER_BACKUP
    void write_backup();
#endif

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();


private:
    Ui::MainWindow *ui;

    ArrowLabel *al1;
    ArrowLabel *al2;
    ArrowLabel *al3;

    QLabel *l1;
    QLabel *l2;
    QLabel *l3;

    QTimer *timer;
    QTimer *sim_timer;

    short throttle;
    short yokeUD;
    short yokeRL;
    short pedal;

#ifdef SERVER_BACKUP
    backup_t bak;
#endif

    SimScreen_GPS *s1;
    SimScreen_EGI *s2;
    InputScreen *s3;

    QPushButton *GPS_switch;
    QPushButton *EGI_switch;

```

```
74      Coordinate_t lat;
75      Coordinate_t lon;
76
77      float altitude;  // feet
78      float heading; // radians
79
80      double speed;
81      speed_t speed_NED;
82      pos_t pos_NED;
83      acc_t acc_NED;
84
85      short pitch, roll, yaw;
86      double yaw_d;
87
88  };
89  #endif // MAINWINDOW_H
```

A.5.1.10 mainwindow.cpp

```
1   #include "mainwindow.h"
2   #include "./ui_mainwindow.h"
3
4   MainWindow::MainWindow(QWidget *parent)
5       : QMainWindow(parent)
6       , ui(new Ui::MainWindow)
7   {
8
9
10      // INIZIALIZZAZIONE
11
12      speed = 0.0;
13
14      throttle = 0;
15      yokeUD = 0;
16      yokeRL = 0;
17
18      // TORINO
19      lat.degree = 45;
20      lat.min = 4;
21      lat.sec = 45;
22
23      lon.degree = 7;
24      lon.min = 40;
25      lon.sec = 34;
26
27      altitude = 0;
28
29      //heading = 0.0f;
30      heading = M_PI/2.0;
31
32      yaw_d = 0.0;
33
34      roll = 0;
35      pitch = 0;
36      yaw = 0;
37
38      l1 = new QLabel("Throttle: ", this);
39      l2 = new QLabel("Yoke: ", this);
40      l3 = new QLabel("Pedal: ", this);
41      l1->setFixedWidth(80);
42      l2->setFixedWidth(80);
43      l3->setFixedWidth(80);
44
45      QLabel *l1_s = new QLabel("[%]", this);
46      QLabel *l2_s = new QLabel("[°]", this);
47      QLabel *l3_s = new QLabel("[°]", this);
48
49      int sat1[4] = {0, 100, 0, 0};
50      int sat2[4] = {-30, 30, -30, 30};
51      int sat3[4] = {0, 0, -30, 30};
52
```

```
53      al1 = new ArrowLabel(5, true, false, sat1, this);
54      al2 = new ArrowLabel(2, true,  true, sat2, this);
55      al3 = new ArrowLabel(5, false, true, sat3, this);
56
57      s1 = new SimScreen_GPS(this);
58      s2 = new SimScreen_EGI(this);
59      s3 = new InputScreen(this);
60
61      QFont font;
62
63      QVBoxLayout *l_v = new QVBoxLayout(this);
64
65      connect(al1, &ArrowLabel::buttonClicked, this, &MainWindow::
        handleArrowLabel);
66      connect(al2, &ArrowLabel::buttonClicked, this, &MainWindow::
        handleArrowLabel);
67      connect(al3, &ArrowLabel::buttonClicked, this, &MainWindow::
        handleArrowLabel);
68
69      connect(this->s3->get_button(), &QPushButton::clicked, this, &MainWindow::
        handleSetButton);
70
71      QWidget *ty_pan = new QWidget(this);
72      QHBoxLayout *throttle_yoke_l = new QHBoxLayout(ty_pan);
73
74      throttle_yoke_l->addWidget(l1);
75      throttle_yoke_l->addWidget(al1);
76      throttle_yoke_l->addWidget(l1_s);
77
78      throttle_yoke_l->addWidget(l2);
79      throttle_yoke_l->addWidget(al2);
80      throttle_yoke_l->addWidget(l2_s);
81
82      throttle_yoke_l->addWidget(l3);
83      throttle_yoke_l->addWidget(al3);
84      throttle_yoke_l->addWidget(l3_s);
85
86      // Creazione dei frame per i widget s1, s2, s3
87      QFrame *frame_s1 = new QFrame(this);
88      frame_s1->setFrameStyle(QFrame::Box | QFrame::Raised);
89      frame_s1->setLineWidth(2); // Spessore del bordo
90      frame_s1->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
        // Politica di espansione
91
92      QFrame *frame_s1_1 = new QFrame(this);
93      frame_s1_1->setFrameStyle(QFrame::Box | QFrame::Raised);
94      frame_s1_1->setLineWidth(1); // Spessore del bordo
95      frame_s1_1->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
         // Politica di espansione
96
97      QFrame *frame_s1_2 = new QFrame(this);
98      frame_s1_2->setFrameStyle(QFrame::Box | QFrame::Raised);
99      frame_s1_2->setLineWidth(1); // Spessore del bordo
100     frame_s1_2->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
         // Politica di espansione
101
102     QFrame *frame_s2 = new QFrame(this);
103     frame_s2->setFrameStyle(QFrame::Box | QFrame::Raised);
104     frame_s2->setLineWidth(2); // Spessore del bordo
105     frame_s2->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
        // Politica di espansione
106
107     QFrame *frame_s3 = new QFrame(this);
108     frame_s3->setFrameStyle(QFrame::Box | QFrame::Raised);
109     frame_s3->setLineWidth(2); // Spessore del bordo
110     frame_s3->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
        // Politica di espansione
111
112
113     // Aggiungi i widget s1, s2, s3 ai loro frame corrispondenti
114     frame_s1->setLayout(new QVBoxLayout(frame_s1));
115
116     frame_s1_1->setLayout(new QVBoxLayout(frame_s1_1));
117     frame_s1_2->setLayout(new QVBoxLayout(frame_s1_2));
118
```

```
    //QLabel *GPS_frame_label = new QLabel("GPS SENSOR", this);
    //font.setBold(true); GPS_frame_label->setFont(font);
    //frame_s1_1->layout()->addWidget(GPS_frame_label);
    GPS_switch = new QPushButton("GPS SENSOR", this);
    GPS_switch->setFixedWidth(100);
    if(s1->get_status()){
        GPS_switch->setStyleSheet(
            "background-color: green; "
            "color: gold; "
            "font-weight: bold; "
            );
    }
    else{
        GPS_switch->setStyleSheet(
            "background-color: red; "
            "color: gold; "
            "font-weight: bold; "
            );
    }

    frame_s1_1->layout()->addWidget(GPS_switch);
    frame_s1_1->layout()->addWidget(s1);

    //QLabel *EGI_frame_label = new QLabel("EGI SENSOR", this);
    //font.setBold(true); EGI_frame_label->setFont(font);
    //frame_s1_2->layout()->addWidget(EGI_frame_label);

    EGI_switch = new QPushButton("EGI SENSOR", this);
    EGI_switch->setFixedWidth(100);
    if(s2->get_status()){
        EGI_switch->setStyleSheet(
            "background-color: green; "
            "color: gold; "
            "font-weight: bold; "
            );
    }
    else{
        EGI_switch->setStyleSheet(
            "background-color: red; "
            "color: gold; "
            "font-weight: bold; "
            );
    }
    frame_s1_2->layout()->addWidget(EGI_switch);
    frame_s1_2->layout()->addWidget(s2);

    QLabel *sensors_frame_label = new QLabel("AIRCRAFT SENSORS", this);

    font.setBold(true); sensors_frame_label->setFont(font);
    frame_s1->layout()->addWidget(sensors_frame_label);

    QWidget *p_h = new QWidget(this);
    QHBoxLayout *l_h = new QHBoxLayout(p_h);
    l_h->addWidget(frame_s1_1);
    l_h->addWidget(frame_s1_2);

    frame_s1->layout()->addWidget(p_h);

    frame_s2->setLayout(new QVBoxLayout(frame_s2));
    QLabel *input_frame_label = new QLabel("POSITION INPUT", this);
    font.setBold(true); input_frame_label->setFont(font);
    frame_s2->layout()->addWidget(input_frame_label);
    frame_s2->layout()->addWidget(s3);


    frame_s3->setLayout(new QVBoxLayout(frame_s3));
    QLabel *command_frame_label = new QLabel("CONSOLE COMMANDS", this);
    font.setBold(true); command_frame_label->setFont(font);
    frame_s3->layout()->addWidget(command_frame_label);
    frame_s3->layout()->addWidget(ty_pan);

    // Aggiungi i frame con i widget al layout principale
    l_v->addWidget(frame_s1);
    l_v->addWidget(frame_s2);
    l_v->addWidget(frame_s3);
```

```
194
195     connect(EGI_switch, &QPushButton::clicked, this, &MainWindow::
        handle_EGI_switch);
196     connect(GPS_switch, &QPushButton::clicked, this, &MainWindow::
        handle_GPS_switch);
197
198     timer = new QTimer(this);
199     sim_timer = new QTimer(this);
200
201 #ifdef SERVER_BACKUP
202     connect(timer, &QTimer::timeout, this, &MainWindow::write_backup);
203     timer->start((int)(delta_T_sec*1000.0));
204 #endif
205
206     connect(sim_timer, &QTimer::timeout, this, &MainWindow::simulate);
207     sim_timer->start((int)(delta_T_sec*1000.0));
208
209     // Configurazione della finestra principale
210     QWidget *centralWidget = new QWidget(this);
211     centralWidget->setLayout(l_v);
212     setCentralWidget(centralWidget);
213
214
215 }
216
217 void MainWindow::handleArrowLabel() {
218
219     throttle = al1->text().toInt();
220     QString y = al2->text();
221     sscanf(y.toStdString().c_str(), "%hd/%hd", &yokeUD, &yokeRL);
222     QString p = al3->text();
223     //short n;
224     sscanf(p.toStdString().c_str(), "%hd", &pedal);
225 }
226
227 void MainWindow::handle_EGI_switch() {
228     if(s2->switch_status()){
229         EGI_switch->setStyleSheet(
230             "background-color: green; "
231             "color: gold; "
232             "font-weight: bold; "
233             );
234     }
235     else{
236         EGI_switch->setStyleSheet(
237             "background-color: red; "
238             "color: gold; "
239             "font-weight: bold; "
240             );
241     }
242 }
243
244 void MainWindow::handle_GPS_switch() {
245     if(s1->switch_status()){
246         GPS_switch->setStyleSheet(
247             "background-color: green; "
248             "color: gold; "
249             "font-weight: bold; "
250             );
251     }
252     else{
253         GPS_switch->setStyleSheet(
254             "background-color: red; "
255             "color: gold; "
256             "font-weight: bold; "
257             );
258     }
259 }
260
261 void MainWindow::handleSetButton(){
262
263     QString latDegText = this->s3->get_lat_deg()->text();
264     if (!latDegText.isEmpty()) {
265         this->lat.degree = latDegText.toInt();
266     }
```

```
267
268     QString latMinText = this->s3->get_lat_min()->text();
269     if (!latMinText.isEmpty()) {
270         this->lat.min = latMinText.toInt();
271     }
272
273     QString latSecText = this->s3->get_lat_sec()->text();
274     if (!latSecText.isEmpty()) {
275         this->lat.sec = latSecText.toInt();
276     }
277
278     QString lonDegText = this->s3->get_lon_deg()->text();
279     if (!lonDegText.isEmpty()) {
280         this->lon.degree = lonDegText.toInt();
281     }
282
283     QString lonMinText = this->s3->get_lon_min()->text();
284     if (!lonMinText.isEmpty()) {
285         this->lon.min = lonMinText.toInt();
286     }
287
288     QString lonSecText = this->s3->get_lon_sec()->text();
289     if (!lonSecText.isEmpty()) {
290         this->lon.sec = lonSecText.toInt();
291     }
292
293
294     QString altText = this->s3->get_alt()->text();
295     if (!altText.isEmpty()) {
296         this->altitude = altText.toInt();
297         this->pos_NED.Z = (FT2MT*altText.toDouble());
298     }
299
300     this->s1->set_all_data(lat, lon, altitude, this->heading, speed);
301
302     kynematics_t kyn;
303     kyn.speed_NED = speed_NED;
304     kyn.speed = speed;
305     kyn.acc_NED = acc_NED;
306
307     this->s2->set_all_data(kyn, this->heading, pitch, roll, yaw, dmsToDec_r(
    lat), dmsToDec_r(lon), altitude);
308
309
310     this->s3->get_lat_deg()->setText("");
311     this->s3->get_lat_min()->setText("");
312     this->s3->get_lat_sec()->setText("");
313
314     this->s3->get_lon_deg()->setText("");
315     this->s3->get_lon_min()->setText("");
316     this->s3->get_lon_sec()->setText("");
317
318     this->s3->get_alt()->setText("");
319
320
321 }
322
323 #ifdef SERVER_BACKUP
324 void MainWindow::write_backup() {
325     //write back_up file
326     time(&(this->bak.timestamp));
327     int fd = open("backup.txt", O_RDWR | O_CREAT | O_TRUNC);
328     write(fd, (void*) &(this->bak), sizeof(this->bak));
329     ::close(fd);
330     this->bak.idx = 0;
331
332
333     // save back_up file into server
334     const char *hostname = "localhost";
335     const char *username = "edoardo";
336     const char *password = "EdoCava00!!";
337
338     int control_sockfd = connect_to_server(hostname, 21);
339     char buffer[BUFFER_SIZE];
340     read_response(control_sockfd, buffer, sizeof(buffer));
```

```
341     //printf("Server response: %s", buffer);
342
343     snprintf(buffer, (6+strlen(username)), "USER %s", username);
344     send_command(control_sockfd, buffer);
345     read_response(control_sockfd, buffer, sizeof(buffer));
346     //printf("%s", buffer);
347
348     snprintf(buffer, (6+strlen(password)), "PASS %s", password);
349     send_command(control_sockfd, buffer);
350     read_response(control_sockfd, buffer, sizeof(buffer));
351     //printf("%s", buffer);
352
353     store_file(control_sockfd, "back_up.txt");
354
355
356     ::close(control_sockfd);
357 }
358 #endif
359
360 MainWindow::~MainWindow()
361 {
362 #ifdef SERVER_BACKUP
363     this->s1->write_backup();
364     this->s2->write_backup();
365 #endif
366     delete ui;
367 }
368
369 int sign(double x) {
370     if (x > 0.0) return 1;
371     if (x < 0.0) return -1;
372     return 0;
373 }
374
375 void MainWindow::simulate(){
376
377     double delta_speed = (MAX_ACCELERATION*(((double)throttle)/150.0)-(
    DRAG_RES*speed/MAX_SPEED))*delta_T_sec;
378     speed += delta_speed;
379
380     if(speed>MAX_SPEED){
381         speed=MAX_SPEED;
382     }
383     if(speed<0.0){
384         speed=MIN_SPEED;
385     }
386
387     if(speed>0){
388         pitch = yokeUD;
389         roll = yokeRL;
390         yaw_d += (double)pedal/2500.0+sign(pedal)*(speed/500000.0)*(
    delta_T_sec/(1.5e-2));
391
392         /*if(yaw_d>180.0){
393             yaw_d-=360.0;
394         }
395         if(yaw_d<-180.0){
396             yaw_d+=360.0;
397         }*/
398
399         // limitations on yaw
400         if(yaw_d>45.0){
401             yaw_d=45.0;
402         }
403         if(yaw_d<-45.0){
404             yaw_d=-45.0;
405         }
406
407         heading += (float) (deg2rad((roll/20000.0)))+sign(roll)*(speed
    /500000.0)*(delta_T_sec/(1.5e-2));
408         if(heading>(float)2.0*M_PI){
409             heading-=(float)(2.0*M_PI);
410         }
411         if(heading<0.0f){
412             heading+=(float)(2.0*M_PI);
```

```
413         }
414
415         yaw = (short) yaw_d;
416
417     }
418
419     speed_t old_speed = speed_NED;
420
421
422     // NED COMPONENTS
423     speed_NED.Z = speed*sin(deg2rad(pitch))*MT2FT;              // feet/s
424     speed_NED.X = speed*cos(deg2rad(pitch))*cos(heading)*MT2FT; // feet/s
425     speed_NED.Y = speed*cos(deg2rad(pitch))*sin(heading)*MT2FT; // feet/s
426
427     acc_NED.X = (speed_NED.X - old_speed.X)/delta_T_sec;
428     acc_NED.Y = (speed_NED.Y - old_speed.Y)/delta_T_sec;
429     acc_NED.Z = (speed_NED.Z - old_speed.Z)/delta_T_sec;
430
431     pos_NED.X += speed_NED.X*delta_T_sec;   // feet
432     pos_NED.Y += speed_NED.Y*delta_T_sec;   // feet
433     pos_NED.Z += speed_NED.Z*delta_T_sec;   // feet
434
435     //printf("POS: %lf %lf\n", pos_NED.X, pos_NED.Y);
436
437
438     if(pos_NED.Z<0.0){
439         pos_NED.Z = 0.0;
440     }
441
442     if(pos_NED.Z>MAX_ALTITUDE){
443         pos_NED.Z = MAX_ALTITUDE;
444     }
445
446     if(pos_NED.Z <= 0.0){
447         if(speed_NED.Z<0.0){
448             speed_NED.Z = 0.0;
449         }
450     }
451
452     // NED -> LAT-LON
453     double c1;
454     double c2;
455     dmsToDec(&c1, lat);
456     dmsToDec(&c2, lon);
457
458     nedToLatLon2(&c1, &c2, speed_NED.X*delta_T_sec*FT2MT, speed_NED.Y*
    delta_T_sec*FT2MT);
459
460     //fflush(stdout);
461
462     decToDMS(c1, &(this->lat));  // update for next step
463     decToDMS(c2, &(this->lon));  // update for next step
464
465     altitude = pos_NED.Z;
466
467     /*this->s1->set_all_data(lat, lon, altitude, heading, speed);
468     kynematics_t kyn;
469     kyn.speed_NED = speed_NED;
470     kyn.speed = speed;
471     this->s2->set_all_data_NO_POS(kyn, heading, pitch, roll, yaw);*/
472
473     //printf("%lf %lf\n", dmsToDec_r(lat), dmsToDec_r(lon));
474
475
476
477     if(speed>0){
478         //printf("%f %f\n", speed_NED.X, speed_NED.Y);
479
480         // GPS
481         double noise = ((double) rand() / RAND_MAX) * 2 * noiseMagnitude -
    noiseMagnitude + 2.0/3600.0;
482         double noise_lat = c1+noise;
483         double noise_lon = c2+noise;
484         noise = ((double) rand() / RAND_MAX) * 2 * noiseMagnitude_h -
    noiseMagnitude_h + speed/250.0;
```

```
485         int noise_alt = altitude+(int)noise;
486 
487         double noise_speed = speed+(((double) rand() / RAND_MAX) * 2 *
    noiseMagnitude - noiseMagnitude);
488         this->s1->set_all_data(decToDMS_r(noise_lat), decToDMS_r(noise_lon),
    noise_alt,
489                                heading, noise_speed);
490 
491         // EGI
492         speed_t noisy_speed_NED = speed_NED;
493         noisy_speed_NED.X+= (((double) rand() / RAND_MAX) * 2 * noiseMagnitude
     - noiseMagnitude);
494         noisy_speed_NED.Y+= (((double) rand() / RAND_MAX) * 2 * noiseMagnitude
     - noiseMagnitude);
495         noisy_speed_NED.Z+= (((double) rand() / RAND_MAX) * 2 * noiseMagnitude
     - noiseMagnitude);
496 
497         noise = ((double) rand() / RAND_MAX) * 2 * noiseMagnitude -
    noiseMagnitude - 2.0/3600.0;
498         noise_lat = c1+noise;
499         noise_lon = c2+noise;
500         noise = ((double) rand() / RAND_MAX) * 2 * noiseMagnitude_h -
    noiseMagnitude_h + speed/250.0;
501         noise_alt = altitude+(int)noise;
502 
503 
504         kynematics_t kyn;
505         kyn.speed_NED = noisy_speed_NED;
506         kyn.speed = speed+(((double) rand() / RAND_MAX) * 2 * noiseMagnitude -
     noiseMagnitude);
507         this->s2->set_all_data_NO_POS(kyn, heading, pitch, roll, yaw);
508     }
509     else{
510         this->s1->set_all_data(lat, lon, altitude, heading, speed);
511 
512         kynematics_t kyn;
513         kyn.speed_NED = speed_NED;
514         kyn.speed = speed;
515         this->s2->set_all_data(kyn, heading, pitch, roll, yaw, c1, c2,
    altitude);
516     }
517 
518 #ifdef SERVER_BACKUP
519     if(this->bak.idx<MAX_RECORDS_NUMBER){
520         this->bak.pos[this->bak.idx].lat = c1;
521         this->bak.pos[this->bak.idx].lon = c2;
522         this->bak.pos[this->bak.idx].alt = altitude;
523         this->bak.att[this->bak.idx].hd = heading;
524         this->bak.att[this->bak.idx].pitch = pitch;
525         this->bak.att[this->bak.idx].roll = roll;
526         this->bak.att[this->bak.idx].yaw = yaw;
527         this->bak.idx++;
528     }
529 #endif
530 
531     this->s1->send_data();
532     this->s2->send_data();
533 
534 }
```

A.5.1.11 main.cpp

```
1 #include "mainwindow.h"
2 
3 #include <QApplication>
4 
5 int main(int argc, char *argv[])
6 {
7     QApplication a(argc, argv);
8     MainWindow w;
9     w.show();
```

```
10      return a.exec();
11  }
```

Appendix B

In this Appendix the structure of the software implementation is reported by means of listing all the files realizing the project, including header, source and configuration files. The appendix is divided into the FACE segments as *sections*. Each *section* is then divided into *subsections* representing the software components, the UoPs or the UoCs inside the segment. In each *subsection*, there are at most three *subsubsections* listing respectively header, source and configuration files if existing.

B.1 PCS

B.1.1 A/C Position Application

B.1.1.1 Header Files

- DF.h

B.1.1.2 Source Files

- DF.cpp
- main.cpp

B.2 TSS

B.2.1 Header Files

- FaceMessageC.h
- FaceMessageTypeSupportS.h
- FaceMessageTypeSupportC.h
- FaceMessage_TS.hpp
- FaceMessageTypeSupportImpl.h
- FaceMessenger_Export.h

B.2.2 Source Files

- FaceMessageTypeSupportC.cpp
- FaceMessage_TS.cpp
- FaceMessageTypeSupportImpl.cpp

B.2.3 Configuration Files

- config_static_DF.ini
- config_static_EGI.ini
- config_static_GPS.ini
- config_static_HM.ini
- config_static_PSGS.ini
- FaceMessageTypeSupport.idl
- FaceMessage.idl
- FaceMessenger_TESI.mpc

B.3 PSSS

B.3.1 Utility Files

B.3.1.1 Header Files

- sim_utils.h

B.3.1.2 Source Files

- sim_utils.cpp

B.3.2 EGI Manager

B.3.2.1 Header Files

- EGI.h
- EGI_manager.h

B.3.2.2 Source Files

- EGI.cpp
- EGI_manager.cpp
- main.cpp

B.3.2.3 Configuration Files

- EGI_ser_monitor_config.cfg
- serialDPM_config.cfg
- serialSIM_config.cfg

B.3.3 GPS Manager

B.3.3.1 Header Files

- GPS.h
- GPS_manager.h

B.3.3.2 Source Files

- GPS.cpp
- GPS_manager.cpp
- main.cpp

B.3.3.3 Configuration Files

- GPS_ser_monitor_config.cfg
- serialDPM_config.cfg
- serialSIM_config.cfg

B.3.4 Health Manager

B.3.4.1 Source Files

- main.cpp

B.3.4.2 Configuration Files

- GPS_ser_monitor_config.cfg
- EGI_ser_monitor_config.cfg

B.3.5 DPM

B.3.5.1 Header Files

- DMP.h
- backup.h

B.3.5.2 Source Files

- DMP.cpp
- main.cpp

B.3.5.3 Configuration Files

- GPS_config.ini
- EGI_config.ini

B.3.6 PSGS Display Application

B.3.6.1 Header Files

- left_screen.h
- right_screen.h
- map_panel.h
- mainwindow.h
- messagereceiver.h

B.3.6.2 Source Files

- left_screen.cpp
- right_screen.cpp
- map_panel.cpp
- mainwindow.cpp
- main.cpp
- messagereceiver.cpp

B.4 IOSS

B.4.1 Utility and Definition Files

B.4.1.1 Header Files

- Common.h
- IO.h
- IO_utils.h

B.4.1.2 Source Files

- IO_utils.cpp

B.4.2 Serial

B.4.2.1 Header Files

- Serial.h
- ConfigFile.h
- MyCallback.h
- serial_constant.h

B.4.2.2 Source Files

- Serial.cpp
- ConfigFile.cpp
- MyCallback.cpp

B.5 A/C Sensors Simulator

B.5.1 Header Files

- arrowlabel.h
- inputscreen.h
- mainwindow.h
- sim_constant.h
- simscreen_egi.h
- simscreen_gps.h

B.5.2 Source Files

- arrowlabel.cpp
- inputscreen.cpp
- mainwindow.cpp
- main.cpp
- simscreen_egi.cpp
- simscreen_gps.cpp

B.5.3 Configuration Files

- serial_EGI_config.cfg
- serial_GPS_config.cfg

Bibliography

[1] The Open Group. *FACE Technical Standard, Edition 3.1*. 2015. URL: https://publications.opengroup.org/standards/face/c207 (cit. on pp. 1, 13–15, 18, 19, 24, 27, 29, 31, 44, 67, 81, 92).

[2] Ansys. URL: https://www.ansys.com/industries/face-standard-aviation-software (cit. on pp. 2, 4).

[3] T. Gaska, C. Watkin, and Y. Chen. "Integrated Modular Avionics - Past, Present and Future". In: (2015) (cit. on pp. 4–6, 8).

[4] R. Ramaker, W. Krug, and W. Phebus. "Application of civil integrated modular architecture to military transport aircraft". In: (2007) (cit. on pp. 4–6).

[5] USDR&E. URL: https://www.cto.mil/sea/mosa/#:~:text=A%20Modular%20Open%20Systems%20Approach,over%20the%20system%20life%20cycle. (cit. on pp. 4, 5).

[6] Christopher B. Watkins. "Integrated Modular Avionics: Managing the Allocation of Shared Intersystem Resources". In: *ieee/aiaa 25TH Digital Avionics Systems Conference* (2006) (cit. on p. 6).

[7] Object Management Group. "OMG Data Distribution Service (DDS)". In: (2015) (cit. on pp. 9, 10).

[8] Object Management Group. URL: https://opendds.org/ (cit. on pp. 9, 10, 28, 44).

[9] Don Busch. URL: https://opendds.readthedocs.io/en/latest-release/index.html (cit. on pp. 10, 13).

[10] Don Busch. URL: https://opendds.org/about/articles/Article-Intro.html (cit. on pp. 10, 11).

[11] Open Group. URL: https://www.opengroup.org/ (cit. on pp. 13, 15).

[12] *Embedded GPS INS*. URL: https://it.wikipedia.org/wiki/Embedded_GPS_INS (cit. on p. 17).

[13] *Global Positioning System*. URL: https://en.wikipedia.org/wiki/Global_Positioning_System (cit. on p. 17).

[14] Inc. RTCA. URL: https://en.wikipedia.org/wiki/DO-178C (cit. on pp. 19, 74).

[15] The Open Group. *Reference Implementation Guide for FACE FACE Technical Standard, Edition 3.0 Vol. 3*. 2015. URL: https://publications.opengroup.org/standards/face/c207 (cit. on pp. 20, 21).

[16] DDC-I. URL: https://www.ddci.com/ (cit. on p. 31).

[17] LYNX. URL: https://www.lynx.com/ (cit. on p. 31).

[18] (Cit. on p. 49).

[19] Military+Aerospace Electronics. "What is safety-certifiable avionics hardware that meets Design Assurance Levels (DAL)?" In: (2016). URL: https://www.militaryaerospace.com/computers/article/16714656/what-is-safety-certifiable-avionics-hardware-that-meets-design-assurance-levels-dal (cit. on p. 74).

[20] *PhenomCINC*. URL: https://www.skayl.com/tools (cit. on p. 82).

[21] *AnsysSCADE*. URL: https://www.ansys.com/it-it/products/embedded-software/ansys-scade-architect (cit. on p. 82).