DOCUMENT CONTROL

Copyright notice

© 2007 European Organisation for the Safety of Air Navigation (EUROCONTROL). All rights reserved.
No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of EUROCONTROL.

Edition history

<table>
<thead>
<tr>
<th>Edition Nº</th>
<th>Effective date or status</th>
<th>Author(s)</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>Draft</td>
<td>SA</td>
<td>First version</td>
</tr>
<tr>
<td>0.2</td>
<td>Draft</td>
<td>SA</td>
<td>Updated according to inputs from MU and NEXT (Industrial advisor)</td>
</tr>
</tbody>
</table>

Acknowledgements

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antonio Monteleone, Senior Engineer at NEXT</td>
<td>NEXT Ingegneria dei Sistemi S.p.A., Rome - Italy</td>
</tr>
<tr>
<td>Yohan Fernando, BSc (Hons) at Middlesex, for his useful work on ATC scenario editing and simulation at MU, from which this work was inspired</td>
<td>Middlesex University</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS

EXECUTIVE SUMMARY .......................................................................................................................... 1

1. INTRODUCTION .................................................................................................................................... 4

2. SYSTEM REQUIREMENTS .................................................................................................................. 6
  2.1 Users and processes .......................................................................................................................... 6
  2.2 Scenario design ............................................................................................................................... 6
  2.3 Experimental run ............................................................................................................................. 7
    2.3.1 Simulation Data .......................................................................................................................... 9

3. PART 2: SOFTWARE ARCHITECTURE ................................................................................................. 11
  3.1 Introduction ....................................................................................................................................... 11
  3.2 Architecture overview ..................................................................................................................... 11
  3.3 Physical architecture ....................................................................................................................... 12
    3.3.1 Simulator architecture ............................................................................................................... 12
      3.3.1.1 Interfaces Description ....................................................................................................... 16
    3.3.2 Editor Architecture .................................................................................................................. 21
      3.3.2.1 Interfaces Description ....................................................................................................... 24
    3.3.3 Data structures .......................................................................................................................... 26
  3.4 Interfaces to external components ................................................................................................... 32
    3.4.1 TCP protocol ............................................................................................................................ 33
    3.4.2 Client - Server communication ................................................................................................. 33

4. PART 3: USER INTERFACE ................................................................................................................ 38
  4.1 Scenario Editor ............................................................................................................................... 38
    4.1.1 Airspace 2D view ....................................................................................................................... 39
    4.1.2 Editor Menu Bar ...................................................................................................................... 40
    4.1.3 Editor Control Bar .................................................................................................................... 41
    4.1.4 Editor Control Panel ............................................................................................................... 41
    4.1.5 Editor Tree-Viewer .................................................................................................................. 42
4.1.6 Creating a Flight Plan ................................................................. 43
4.1.7 Adding Weather Objects ......................................................... 45
4.1.8 Adding Scenario Complementary Information ............................. 46

4.2 Scenario Player ........................................................................ 47
  4.2.1 Airspace 2D view ................................................................. 48
  4.2.2 Simulator Menu Bar ............................................................... 49
  4.2.3 Simulator Control Bar ............................................................. 50

APPENDIX A – SOFTWARE CODE DOCUMENTATION .......................... 51
**EXECUTIVE SUMMARY**

This document represents the third deliverable (D1.3) of the INO 2D 3D project, sponsored by EUROCONTROL. The scope of the project is to define advanced visualization designed to improve the controllers’ job, by exploiting the benefits of 2D and 3D visualization.

As part of the project, a software infrastructure has been developed to enable the project investigators to easily create dynamic air traffic scenarios and use them to test the different 3D/2D HMI concept prototypes that are supposed to emerge during the project.

Such a software infrastructure is also referred to as Experimental Testbed and basically consists of an editing facility allowing the experimenters to create scenarios (typically including waypoints, fixes, weather and flying airplanes) and a simulation environment (simulation engine) capable to run those scenarios, to interface to the different 3D-in-2D HMI prototypes, which will display the simulated traffic in real-time, and also to support the measurements of the user response.

This document provides a description of this software, including the user requirements that have informed the design, the software architecture (including data structures, physical components and interfaces), and the user interface for accessing the software capabilities and running the experiments.
LIST OF FIGURE

Figure 1. Controller, Pseudo-Pilot and scenario simulator: data-flow logic for traffic control........... 7
Figure 2. Global Architecture Overview.................................................................................. 11
Figure 3. UML diagram showing the simulator architecture.................................................. 12
Figure 4. UML diagram showing the Editor architecture....................................................... 21
Figure 5. Client-server communication scheme................................................................. 32
Figure 6. Scenario Editor: overview ....................................................................................... 37
Figure 7. Scenario Editor: airspace 2D view ......................................................................... 38
Figure 8. Scenario Editor: conflict / intersections visualisation ......................................... 39
Figure 9. Scenario Editor: File menu..................................................................................... 39
Figure 10. Scenario Editor : Scenario menu ................................................................. 40
Figure 11. Scenario Editor: time control slider..................................................................... 40
Figure 12. Scenario Editor: aircraft move as the time slider is shifted ............................. 40
Figure 13. Scenario Editor – Control panel .......................................................................... 41
Figure 14. Scenario Editor: airspace tree-viewer ............................................................... 42
Figure 15. Scenario Editor: Dialog for adding a Fix......................................................... 43
Figure 16. Scenario Editor: adding random Fixes ............................................................... 43
Figure 17. Scenario Editor: setting up a flight plan............................................................. 44
Figure 18. Scenario Editor: creating a flight plan............................................................... 44
Figure 19. Scenario Editor: adding a new weather object ............................................... 45
Figure 20. Scenario Editor: adding textual comments to a scenario............................... 46
Figure 21. Simulator: user interface overview ................................................................ 47
Figure 22. Simulator: airspace 2D view.............................................................................. 48
Figure 23. Simulator: File menu ....................................................................................... 48
Figure 24. Simulator: simulation menu............................................................................... 49
Figure 25. Simulator: control bar....................................................................................... 49
1. INTRODUCTION

This document represents the third deliverable (D1.3) of the INO 2D 3D project, sponsored by EUROCONTROL. The scope of the project is to define advanced visualisation designed to improve the controllers’ job, by exploiting the benefits of 2D and 3D visualization.

As part of the project, a software infrastructure has been developed to enable the project investigators to easily create dynamic air traffic scenarios and use them to test the different 3D/2D HMI concept prototypes that are supposed to emerge during the project.

Such a software infrastructure is also referred to as Experimental Testbed and basically consists of an editing facility allowing the experimenters to create the scenarios (typically including waypoints, fixes, weather and flying airplanes) and a simulation environment (simulation engine) capable to run those scenarios, to interface to the different 3D-in-2D HMI prototypes, which will display the simulated traffic in real-time, and also to support the measurements of the user response.

The project team has chosen to develop such a platform with the aim of having an easy to use tool for creating relatively simple air traffic scenarios for HMI testing purposes, with special emphasis on evaluation (at least for year 1 of the project) of visuo-spatial aspects and expected benefits of highly innovative 3D-in-2D HMI concepts. For this reason, simplicity and ease of use have been considered as important points.

A possible alternative solution could have been to adopt existing ATC simulation platforms, such as ACE, ESCAPE or eDEP (from Eurocontrol) platforms. Those platforms would certainly provide the possibility of creating much more realistic ATC simulation scenarios; however, such complexity and realism has been found beyond the scope of the early phases of the project, where the focus is more on evaluation of innovation rather than on operationally realistic types of assessment.

On the other hand, the possibility to migrate to existing, standard simulation platform will be considered by the project consortium as the project will move on (in year 2 and especially year 3) in the perspective of an integration of the innovative 3D-in-2D technology in the industrial/operational context.

This document provides a description of the Testbed software, including the user requirements that have informed the design, the software architecture (including data structures, physical components and interfaces), and the user interface for accessing the software capabilities and running the experiments.

The document is split in the following main parts.

- Chapter 1 is this introduction.
- Chapter 2 provides an overview of the requirements from the user point of view.
- Chapter 3 provides the technical description of the software architectural aspects, including the high-level functional architecture and the physical architecture with the main software components, data structures and interfaces. In particular, the communication interface for data exchange between the simulation software and the HMI prototypes is described in this section.
- Chapter 4 describes the user interface of the software, for creating and editing the air traffic scenarios as well as for simulating (executing) them and connecting the simulator to the HMI prototypes that the user wants to evaluate.
- Appendix A includes the software code documentation.
2. SYSTEM REQUIREMENTS

This section describes the user requirements for the Experimental Testbed software.

The requirements were captured through meetings and discussion between consortium partners (SA and MU) and are mainly based on experience on past projects in the ATC domain where both partners were involved.

2.1 Users and processes

Three categories of users of the Experimental Testbed have been identified:

- the HMI investigators, also referred to as ‘experimenter’, whose role is to create air traffic scenarios, setup their simulation and supervise the experiments for testing the different HMI prototypes;
- the experiments’ subjects, playing the role of air traffic controllers and who are required to interface to the simulated scenarios through the HMI prototypes during the experiment execution;
- the pseudo-pilots, who interact with the subjects (controllers) usually via voice communication and are responsible for controlling selected aircraft in the simulated scenarios (according to instructions / requests coming from controllers).

The main processes in which these users are involved are:

- **scenario design**, consisting in the preparation of one or more air traffic scenarios using dedicated software tools;
- **experiment run**, consisting in the simulation of the scenarios for testing the different HMI prototypes and in collecting data for measuring the subject’s response.

The following sections cover the system requirements, given from the user point of view, for the air traffic scenario design process as well as for the experiment execution.

2.2 Scenario design

The scenario design process shall allow the users (experimenters) to set the air traffic conditions that the subjects will play with during the execution of the experiment.

The only user involved in this project is the experimenter.

When creating a scenario, the experimenter shall be able to perform the following operations:

- specify a geographic region containing the airspace of the scenario under creation; the geographic region will be defined by the user as a rectangular 2D region (e.g. expressed by latitude and longitude boundaries, or equivalent representation);
- create a set of waypoints (“fixes”), by inputting the position and name (code) of each new waypoint;
- ask the system to automatically generate a number of randomly-located waypoints;
- manually create aircraft flight plans; for each flight plan, the user will be able to enter the following information:
o the 2D trajectory over a given geographic region, specified as a sequence of waypoints;
o the start time of the flight, expressed in hours, minutes and seconds;
o the flight level at which the aircraft will fly, with the possibility to set either a constant value for the whole path or a different flight level for each waypoint;
o the speed at which the aircraft will fly, with the possibility to set either a constant value for the whole path or a different speed for each waypoint;

• create simple weather objects, like e.g. a cumulonimbus or a turbulent region in the airspace; the weather objects could be defined as a simple volume region within the airspace;
• verify the presence of potential events in the scenario, such as conflicts between airplanes (loss of separation) or airplanes crossing a weather object, based on the flight-plans and the weather objects specified; events should be described (at least) by their type, time of occurrence and location; the system should support the automatic detection of these events and notify them to the user (e.g. by displaying them at their geographic location in the scenario environment);
• ask the system to simulate the scenario and visualise (with a simple 2D representation) the traffic evolution, so that the user can visually assess the scenario which is being created, i.e. identifying the criticalities, the events, the level of complexity (qualitative assessment), so that the user can refine it iteratively;
• implement time line control (e.g. slider) to monitor how the scenario would change in time and refine it;
• play / preview the scenario in 2D (nice to have: preview with 3D view)
• save a scenario, including all the information related to the airspace, the flight plans, the weather objects and the detected scenario events;
• load a previously created scenario, in order to edit/modify it;
• insert a textual description or comment and associate it to the current scenario;
• specify the level of difficulty of each scenario, by inputting a score/rating (this score shall be saved with the corresponding scenario).

User Interface

In addition to the requirements above, the scenario editing software shall implement a graphical user interface (GUI) including at least the following elements:

- a window displaying a simple 2D representation of the airspace, including fixes with labels, flight plans (paths) and weather objects;
- menus and panels with buttons to access the scenario editing functions;
- dialog boxes enabling the easy creation of the scenario objects (flight plans, clouds, etc.).

The GUI will support the possibility to create the paths of the flight plans through direct mouse interaction with the 2D airspace representation, i.e. selecting and connecting waypoints by mouse-clicking on them.
2.3 Experimental run

During the experimental run, the experimenter and the subject will be both involved; the former will be responsible for setting up the scenarios, the simulation engine and the HMI prototype to be tested, while the latter will be required to control the running air traffic scenario using the 3D-2D HMI selected by the experimenter.

It must be noted that the subject could be either an air traffic controller or a non-domain expert, such as students. With the former it would be possible look at operational aspects, with the latter more at Perceptual aspects, with simplified traffic scenarios.

In addition to these main actors, one pseudo-pilot will be needed for controlling specific aircraft according to the instructions raised by the controller (= the subject). Typically, instructions are communicated by the controller to the pseudo-pilot via voice (e.g. via radio), as would occur between a controller and an aircraft pilot in a real operational context. The pseudo-pilot will perform the instructions on the aircraft under control (e.g. change speed, flight level, direction, etc.) accordingly, through a dedicated user interface. Figure below shows a high-level description of the overall data flow, involving the controller, the pseudo-pilot and the ATC scenario simulation software.

Figure 1. Controller, Pseudo-Pilot and scenario simulator: data-flow logic for traffic control.

User: Experimenter

During the experiment setup process (after scenario creation), the experimenter shall to be able to do the following:

- load a scenario, choosing among the ones previously created in the scenario editing process;
- setup the simulator so that it is ready to connect/interface to a 3D-2D HMI prototype (ideally, this should be automatically done by the system after loading a scenario);
- choose the 3D-2D HMI prototype to test;
- launch and connect the HMI prototype to the scenario simulator;
- start the simulation of the scenario.

Under normal conditions, once started, the experimental scenario should run without pauses or interruptions; however, the experimenter should be able to control the simulation in order to cope
with special circumstances or unexpected situations that may require for instance the pausing or 
restarting of the simulation.

Therefore, during the experimental run, the experimenter should be able to:
- pause the scenario simulation;
- restore the scenario simulation after a pause;
- stop and restart the scenario simulation;
- accelerate the time-speed of the simulation execution (with respect to the ‘real-time’ 
speed).

User: Subject of the experiment

During the experimental run, the subject (controller) shall to be able to do the following:

- start scenario (if not already done by the experimenter);
- use the 3D-2D HMI (this requirement however does not apply to the simulator itself);
- input human data response;
- assign instructions to aircraft, via communication to the pseudo-pilot.

Subject performance measurement (Advanced Version of the simulator, year 2) shall include the 
collection, measurement and recording of (at least) the following:
- subject response time to a scenario event (e.g. a conflict);
- subject stress level/workload, including the possibility to create distraction exercises to 
increase the workload;
- situational awareness.

User: Pseudo Pilot (Advanced Version of the simulator, year 2)

During the experimental run, the pseudo-pilot shall to be able to do the following:

- communicate with controller;
- assign instructions to aircraft via a dedicated user interface to the simulator.

2.3.1 Simulation Data

During an experiment run, the simulation core (simulator) of the Testbed software should exchange 
data with the 3D-2D HMI in order to make the necessary scenario information available to the user 
(controller); the following data should be sent by the simulator:
- Updated position of each aircraft (e.g. latitude and longitude or equivalent information);
- updated flight level of each aircraft;
- Fixes data, such as identification code, type and position of each Fix;
- Flight-plans data, including the list of waypoints of each flight plan, the start time, the 
planned speed and flight levels along the flight plans;
- Data on “weather objects” (i.e. airspace volumes where a cumulonimbus or turbulence region is located) and other restricted airspace volumes, including position, horizontal and vertical extensions of such volumes.

**Requirements for Year 2 of the project (Advanced Version):**

Some of the above requirements will be supported by the advanced version of the simulator to be developed in the 2nd year of the project.

In particular, the following requirements do not apply to the current version of the experimental testbed:

- the support of the pseudo-pilot interface;
- the possibility to collect and measure subject performance and response during scenario execution.

During Year 2 of the project, requirements will also be defined more precisely and the simulator/testbed infrastructure will be updated accordingly.
3. PART 2: SOFTWARE ARCHITECTURE

3.1 Introduction

This section describes the software architecture of the 3D-in-2D Experimental Testbed. In the following subsections the overall architecture of the system is presented, followed by a more detailed description of the software components and their relationships and interfaces. The main data structures used in the software are also described.

Finally, the interfaces to external components, typically consisting of the 3D-in-2D HMI prototypes developed in the project, will be introduced and explained in detail.

3.2 Architecture overview

The Experimental Testbed consists of the following main modules:
- the Simulator;
- the ATC Scenario Editor;
- the Pseudo-Pilot Interface (to be implemented in Year 2 of the project).

Such decomposition has been adopted because it reflects the different processes (scenario editing, experiment run, communication with pseudo-pilot) identified in the user requirements.

In figure below an overview of the main modules is shown. Beside the modules listed above, the 3D-in-2D HMI is represented as well, although it is not part of the experimental testbed (i.e. it has to be considered as an external module).

The Scenario Editor is the module by which an experimenter can create the ATC scenarios that will be then executed during the experiment.

The scenarios created are stored in the Scenario DB, from which the Simulator can retrieve the scenario data for setting up and running the simulation.

The Simulator exchanges data and events with the 3D-2D HMI, sending simulation data and receiving inputs (subject response) from the user (for instance, an input could consist in a mouse-click of the subject to select an aircraft in the HMI).
In particular, the data sent by the Simulator module to the 3D-2D HMI (and to the Pseudo-Pilot Interface for the Advanced version to be developed in year 2 of the project), referred to as “Sim Data” in figure 2, reflect the ones specified in the requirements, i.e. including:

- Updated position of each aircraft (e.g. latitude and longitude or equivalent information);
- Updated flight level of each aircraft;
- Fixes data (identification code, type and position of each Fix);
- Flight-plans data, including the list of waypoints of each flight plan, the start time, the planned speed and flight levels along the flight plans;
- Data on “weather objects” (i.e. airspace volumes where a cumulonimbus or turbulence region is located) and other restricted airspace volumes, including position, horizontal and vertical extensions of such volumes.

The data exchange between the Simulator and the 3D-2D HMI is achieved through a Client-Server communication protocol (TCP-based), as it will be explained in section 3.4 of this document. Please refer to the section 3.4.2 (“Client-Server Communication”) for details on the precise type of exchanged data.

### 3.3 Physical architecture

This section discusses the physical architecture, explaining the software components constituting it and their main interfaces.

#### 3.3.1 Simulator architecture

The simulator is the module in charge of executing an ATC scenario previously created with the Editor.
The following figure shows a physical diagram illustrating the simulator components and their relationships.

![UML diagram showing the simulator architecture.](image)

**Figure 3. UML diagram showing the simulator architecture.**

In this diagram we also show the user, interfacing to the Simulator through the *SimUI* component, i.e. the graphical user interface, and the 3D-in-2D HMI software module which acts as a client in a TCP-IP connection and is interfaced to the *SimServer* component of the simulator.

The architectural diagram includes the following components:
In the following, a description of each component and its interfaces and dependencies from other components is given.

Each interface typically corresponds to a method or a group of methods. Methods are implemented as member functions of Java classes. An interface can represent either a way to receive some specific data from the component or otherwise to send a command to the component.

**SimManager**

This component is responsible for guiding the overall execution of the simulator, responding to requests coming from the user interface, updating the airspace and scenario data. It is interfaced to almost all major components of the simulator.

It exposes the following interfaces ():

- **AirspaceData**: used for accessing the data that are managed by the SimManager;
- **SimAirplanesData**: used for retrieving updated data of the simulated airplanes;
- **SimControl**: used for controlling the execution of the simulation, i.e. to start, pause, stop, resume the simulation;
- **LoadScenario**: used to command the manager to activate the process for loading a scenario into the simulator;

The SimManager depends on the following components:

- **SimServer**, for enabling the TCP server connection;
- **SimEngine**, for the actual update of aircraft position, speed and other data, as well as for conflict detection aspects;
- **ATCFFileIO**, for loading airspaces and/or scenarios;

**SimUI**

This component is responsible for providing a graphical interface to the user and managing the user inputs. Its main functions include the setting of display options in the airspace 2D renderer and forwarding user commands to the SimManager.

It depends on the following components:

- **SimAirspaceRenderer**, for setting the visual options used to render the airspace (e.g. "show Fix labels", etc.);
- **SimManager**, for executing the user requests.
**SimAirspaceRenderer**

This component is responsible for rendering the 2D view of the airspace, including flight-plans trajectories, fixes, airplanes and weather objects.

It exposes the following interface:

- **SetVisualOptions**, for setting the display options of the renderer, e.g. to specify which data layers are to be visualised;

The **SimAirspaceRenderer** depends on the following components:

- **SimManager**, for retrieving updated data on the airspace and the simulated airplanes.

**SimServer**

This component is responsible for setting up a TCP server and managing the connection with a client, exchanging data with it.

It exposes the following interfaces:

- **ServerControl**, used for controlling the start and stop of the server itself;
- **SimData**, used by a client to exchange data with the server.

The **SimServer** depends on the following components:

- **SimManager**, for accessing updated airspace and simulated aircraft data.

**SimEngine**

This component is responsible for simulating the scenario dynamics. In particular, it is in charge of updating the state (position, altitude, speed, direction) of all the airplanes based on their flight-plans and according to the temporal execution settings, i.e. taking into account if the simulation is paused and the current "time speed" settings (real time or accelerated time).

This component is also responsible for managing the detection of conflicts occurring between aircraft or intersections of an aircraft with a weather object. However, to this aim it uses the services provided by two dedicated software components.

Finally, the **SimEngine** component is also in charge of recording the significant events occurring during the simulation, i.e. the conflicts between airplanes and the intersections between an aircraft and a weather object; such events are recorded in a log file ("simlog.txt") on the local file system.

This component exposes the following interfaces:

- **SimControl**, used for controlling the execution of the simulation, i.e. to start, pause, stop, resume the simulation;
- **SimAirplanesData**, used for accessing data on simulated aircraft;
- **ConflictsData**, used for accessing data on detected conflicts and aircraft-cloud intersections.

The **SimEngine** depends on the following components:

- **AircraftConflictFinder**, for conflict detection;
- **AircraftCloudIntersectionFinder**, for detection of intersections between an aircraft and a weather object.

**AircraftConflictFinder**
This component is responsible for detecting conflicts between aircraft moving in the airspace during the simulation.

It exposes the following interfaces:
- ConflictsData, used for accessing the data representing the conflicts detected;
- CheckConflicts, used to command the component to execute the detection of possible conflicts;

The AircraftConflictFinder depends on the SimEngine component, for retrieving the data of the simulated airplanes (on which to apply the conflict detection algorithms).

**AircraftCloudIntersectionFinder**

This component is responsible for detecting intersections between aircraft and weather objects (“clouds”) during the simulation.

It exposes the following interfaces:
- IntersectionData, used for accessing the data representing the intersections detected;
- CheckIntersections, used to command the component to execute the detection of possible aircraft-cloud intersections;

The AircraftConflictFinder depends on the SimEngine component, for retrieving the data of the simulated airplanes (on which to apply the intersection detection algorithms).

**ATCFileIO**

This component is responsible for loading and saving scenarios from/to the file system. This component is used by both the Simulator and the Editor. Each scenario is created with the Editor module and conveniently saved in an XML file which can then be loaded by the Simulator.

When loading a scenario, the ATCFileIO reads the XML scenario file, parses the XML data structure and builds a scenario data structure in the system memory. In particular, a scenario is formed by the combination of airspace data and complementary information (textual comments and level of difficulty of the scenario) entered by the user at scenario editing time.

When saving a file, the ATCFileIO component translates the scenario data structure into a DOM (Document Object Module) data structure and finally writes the XML file according to such a structure.

In the case of the Simulator, this component is only used for loading scenarios; the saving capabilities are not used by the simulator, however they are used in the Editor module, where the same component is used.

The component exposes the following interface:
- ScenarioData, for retrieving the data describing the airspace and the complementary scenario info;
- ScenarioReadWrite, used for activate the process of loading and saving a scenario from the file system (in the case of the Simulator, only the loading capability is used).
3.3.1.1 Interfaces Description

In this section, for each interface identified in the physical architecture, a description is provided, including the operations supported by each interface (implemented by one or more class-methods) and the data structures used (refer to section 3.3.3 for a detailed description of each data structure).

<table>
<thead>
<tr>
<th>Component: SimManager</th>
<th>Interface: AirspaceData</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>The AirspaceData interface retrieves and returns the data describing an airspace and its objects (fixes, flight plans, weather objects, etc); it includes the following methods:</td>
</tr>
<tr>
<td></td>
<td>• getAirspace(), returning a pointer to an &lt;Airspace&gt; object describing the airspace associated to the current scenario;</td>
</tr>
<tr>
<td>Data structures involved:</td>
<td><strong>Airspace</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component: SimManager</th>
<th>Interface: LoadScenario</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>The LoadScenario interface of the SimManager is used to command the loading of a scenario from file; it includes the following methods:</td>
</tr>
<tr>
<td></td>
<td>• loadScenario(), loads a scenario from a specified file;</td>
</tr>
<tr>
<td>Data structures involved:</td>
<td><strong>NONE</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component: SimManager</th>
<th>Interface: SimControl</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>The SimControl interface allows to control the main simulation timing parameters, through the following methods:</td>
</tr>
<tr>
<td></td>
<td>• setTimeSpeed(), to set the speed of the simulation with respect to real time;</td>
</tr>
<tr>
<td></td>
<td>• startSim(), to start the simulation, i.e. the update process of the simulated airplanes in the scenario;</td>
</tr>
<tr>
<td></td>
<td>• stopSim(), to stop the simulation and reset the simulation time to the start time;</td>
</tr>
<tr>
<td></td>
<td>• resumeSim(), to resume the simulation which had been paused;</td>
</tr>
<tr>
<td></td>
<td>• pauseSim(), to pause the simulation;</td>
</tr>
</tbody>
</table>
## Data structures involved: NONE

### SimManager
**Interface:** SimAirplanesData
**Description:**
The SimAirplanesData interface retrieves and returns the updated data on the simulated airplanes in the scenario; the data are returned as a list of SimAircraft objects, through the following method:
- `getSimAirplanes();`

### SimEngine
**Interface:** ConflictsData
**Description:**
The ConflictsData interface retrieves and returns the data describing the conflicts between aircraft having intersecting trajectories and the intersections between aircraft and weather objects (clouds); it includes the following methods:
- `getConflicts()`, returning a pointer to a list of conflicts (aircraft-aircraft and intersections aircraft-cloud) detected by the simulation engine;

### SimEngine
**Interface:** SimControl
**Description:**
The SimControl interface allows to control the main simulation parameters, through the following methods:
- `setTimeSpeed()`, to set the speed of the simulation with respect to real time;
- `start()`, to start the simulation, i.e. the update process of the simulated airplanes in the scenario;
- `stop()`, to stop the simulation and reset the simulation time to the start time;
- `rewind()`, to bring the simulation time back to initial conditions (start time) and put the simulation in paused mode;
- `play()`, to put the simulation in "play mode" (note: a `start()` command is equivalent to a `rewind()` followed by a `play()`);
- `enableConflictCheck()`, to enable the conflict detection during the simulation;
• `disableConflictCheck()`, to disable the conflict detection.

Data structures involved: NONE

<table>
<thead>
<tr>
<th>Component: <strong>SimEngine</strong></th>
<th>Interface: <strong>SimAirplanesData</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>The SimAirplanesData interface retrieves and returns the updated data on the simulated airplanes in the scenario; the data are returned as a list of <code>&lt;SimAircraft&gt;</code> objects, through the following method:</td>
</tr>
<tr>
<td></td>
<td>• <code>getSimAirplanes()</code>;</td>
</tr>
<tr>
<td>Data structures involved:</td>
<td><strong>SimAircraft</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component: <strong>AircraftConflictFinder</strong></th>
<th>Interface: <strong>CheckConflicts</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>The CheckConflicts interface allows enabling the detection of aircraft conflicts through the following method:</td>
</tr>
<tr>
<td></td>
<td>• <code>checkConflicts()</code>;</td>
</tr>
<tr>
<td>Data structures involved:</td>
<td>NONE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component: <strong>AircraftConflictFinder</strong></th>
<th>Interface: <strong>ConflictsData</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>The ConflictsData interface of the AircraftConflictFinder retrieves and returns the data describing the conflicts found between aircraft having intersecting trajectories (does <strong>not</strong> include aircraft-cloud intersections); it includes the following methods:</td>
</tr>
<tr>
<td></td>
<td>• <code>getConflicts()</code>, returning a pointer to a list of <code>&lt;AircraftConflict&gt;</code> objects, representing the conflicts detected;</td>
</tr>
<tr>
<td>Data structures involved:</td>
<td><strong>AircraftConflict</strong></td>
</tr>
</tbody>
</table>
### Component: AircraftCloudIntersectionFinder | Interface: CheckIntersections

**Description:**
The CheckIntersections interface allows enabling the detection of aircraft-cloud intersections through the following method:
- `checkIntersections();`

**Data structures involved:** NONE

---

### Component: AircraftCloudIntersectionFinder | Interface: IntersectionsData

**Description:**
The IntersectionsData interface of the AircraftConflictingFinder retrieves and returns the data describing the intersections of an aircraft with a weather object (cloud); it includes the following methods:
- `getIntersections()`, returning a pointer to a list of `<AircraftCloudIntersection>` objects representing the intersections detected;

**Data structures involved:** AircraftCloudIntersection

---

### Component: ATCFileIO | Interface: ScenarioData

**Description:**
The ScenarioData interface retrieves and returns the data describing the scenario and airspace loaded from file by the ATCFileIO component; it includes the following methods:
- `getAirspace()`, returning a pointer to the loaded airspace data structure;
- `getScenario()`, returning a pointer to the loaded scenario data structure.

**Data structures involved:** Scenario, Airspace

---

### Component: ATCFileIO | Interface: ScenarioReadWrite

**Description:**
The ScenarioReadWrite interface allows loading and saving XML files containing the scenario and airspace data; it includes the following methods:
- `readScenarioFile()`, to load a scenario from file;
- `writeScenarioFile()`, to save a scenario in a file;
- `readAirspaceFile()`, to load an airspace from file;
- `writeAirspaceFile()`, to save an airspace in a file;

Data structures involved: **Scenario, Airspace**

### Component: **SimServer**  
**Interface:** **ServerControl**

**Description:**
The `ServerControl` interface allows to set up the server for accepting and establishing connections from clients, through the following methods:
- `listen()`, to set up the server and put it in “listening mode” (waiting for a client; when a client connection request is received, the connection is established and the server starts working);
- `sendLine()`, to send data to the client as a text string.

Data structures involved: **NONE**

### Component: **SimServer**  
**Interface:** **SimData**

**Description:**
The `SimData` interface is used to exchange data with the client HMI; it includes the following methods:
- `read()`, to read the messages (requests) from the Client and respond to it (through the TCP communication) with the requested data.

Data structures involved: **NONE**

### Component: **SimAirspaceRenderer**  
**Interface:** **SetVisualOptions**

**Description:**
The `SetVisualOptions` interface allows to set up the graphical settings of the simulator main view-port, through the following methods:
- `setGridColor()`, to set the colour of the background grid;
- `setGridSpacing()`, to set the spacing of the background grid;
- `showFixLabels()`, to enable/disable the visualisation of Fix labels;
Data structures involved: NONE

3.3.2 Editor Architecture
The Editor is the software module which allows the user to create, edit and store ATC scenarios. The following figure shows a physical diagram illustrating the Editor components and their relationships.
Figure 4. UML diagram showing the Editor architecture.

In this diagram we also show the user, interfacing to the Editor through the EditorUI component, i.e. the graphical user interface.

The architectural diagram includes the following components:

- EditorManager;
- EditorUI;
• EditorAirspaceRenderer;
• SimEngine;
• AircraftConflictFinder;
• AircraftCloudIntersectionFinder;
• ATCFileIO.

In the following, a description of each component and its interfaces and dependencies from other components is given.

Each interface typically corresponds to a method or a group of methods. Methods are implemented as member functions of Java classes. An interface can represent either a way to receive some specific data from the component or otherwise to send a command to the component.

**EditorManager**

This component is responsible for guiding the overall execution of the editor, responding to requests coming from the user interface and updating the airspace and scenario data accordingly. It is interfaced to almost all major components of the Editor.

It exposes the following interfaces ():

• AirspaceData: used for accessing the data that are managed by the EditorManager;
• SimAirplanesData: used for retrieving updated data of the simulated airplanes;
• EditScenario: used for translating the user inputs into actual editing operations applied to the scenario and airspace;
• LoadSaveScenario: used to command the manager to activate the process of loading or saving a scenario into the Editor;

The EditorManager depends on the following components:

• SimEngine, for the actual update of aircraft position, speed and other data, as well as for conflict detection aspects;
• ATCFileIO, for loading and saving airspaces and/or scenarios;

**EditorUI**

This component is responsible for providing a graphical interface to the user and managing the user inputs. Its main functions include the setting of display options in the airspace 2D renderer and forwarding user commands to the EditorManager.

It depends on the following components:

• EditorAirspaceRenderer, for setting the visual options used to render the airspace (e.g. "show Fix labels", etc.);
• EditorManager, for executing the user requests.

**EditorAirspaceRenderer**
This component is responsible for rendering the 2D view of the airspace, including flight-plans trajectories, fixes, airplanes and weather objects.

It exposes the following interface:

- SetVisualOptions, for setting the display options of the renderer, e.g. to specify which data layers are to be visualised;

The EditorAirspaceRenderer depends on the following components:

- EditorManager, for retrieving updated data on the airspace and the simulated airplanes.

SimEngine

This component is responsible for simulating the scenario dynamics.

This component is the same used in the Simulator module. Details can be found in the Simulator architecture sections.

In the context of the Editor, the SimEngine component is used with a different aim than in the Simulator module; in fact, the Editor uses this component not for real time simulation purposes, but rather for the following reasons:

- to enable a quick preview of the scenario simulated dynamics; in particular, it allows the user to interactively set the time along the scenario timeline and see the resulting change of position of all the aircraft in the scenario; this function is used in order to assess the scenario evolution while creating/editing it;
- to preliminarily detect potential conflicts and aircraft-cloud intersections, based on the flight-plans and airspace configuration; this function is useful to assess the level of difficulty of a scenario before the actual simulation runs.

AircraftConflictFinder

This component is the same used in the Simulator module. Details can be found in the Simulator architecture sections.

AircraftCloudIntersectionFinder

This component is the same used in the Simulator module. Details can be found in the Simulator architecture sections.

ATCFileIO

This component is responsible for loading and saving scenarios from/to the file system.

This component is the same used in the Simulator module. Details can be found in the Simulator architecture sections.

However, in the case of the Editor, both the loading and saving capabilities are used (in the Simulator, only the loading function was used).
### 3.3.2.1 Interfaces Description

In this section, for each interface identified in the physical architecture, a description is provided, including the operations supported by each interface (implemented by one or more class-methods) and the data structures used (refer to section 3.3.3 for a detailed description of each data structure). Because some of the components also appear in the Simulator architecture described in the previous section 3.3.1, this section only includes the interfaces which are specific to the Editor (refer to section 3.3.1.1 for the interfaces of components shared with the Simulator).

<table>
<thead>
<tr>
<th>Component: EditorManager</th>
<th>Interface: LoadSaveScenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td></td>
</tr>
<tr>
<td>The LoadSaveScenario interface of the SimManager is used to command the loading and saving of a scenario from/to file; it includes the following methods:</td>
<td></td>
</tr>
<tr>
<td>• loadScenario(), loads a scenario from file;</td>
<td></td>
</tr>
<tr>
<td>• saveScenario(), saves a scenario on file;</td>
<td></td>
</tr>
<tr>
<td>• loadAirspace(), loads an airspace from file;</td>
<td></td>
</tr>
<tr>
<td>• saveAirspace(), saves an airspace to file.</td>
<td></td>
</tr>
</tbody>
</table>

Data structures involved: **Airspace, Scenario**

<table>
<thead>
<tr>
<th>Component: EditorManager</th>
<th>Interface: EditScenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td></td>
</tr>
<tr>
<td>The EditScenario interface allows to make a number of operations related to the editing of a scenario; it includes the following methods:</td>
<td></td>
</tr>
<tr>
<td>• selectFPlan(), to mark a flight plan (identified by name) as selected, so that subsequent editing operations apply to it;</td>
<td></td>
</tr>
<tr>
<td>• selectCloud(), to mark a cloud as selected, so that subsequent editing operations apply to it;</td>
<td></td>
</tr>
<tr>
<td>• selectFix(), to mark a Fix as selected, so that it is highlighted in the editor main view;</td>
<td></td>
</tr>
<tr>
<td>• startCreatingPlan(), to initialize the creation procedure for a new flight plan;</td>
<td></td>
</tr>
<tr>
<td>• abortCreatingPlan(), to cancel the current flight-plan creation process;</td>
<td></td>
</tr>
<tr>
<td>• endCreatingPlan(), to finalize the creation process of a flight plan;</td>
<td></td>
</tr>
<tr>
<td>• removePlan(), to remove a flight plan (the selected one, see also selectFPlan());</td>
<td></td>
</tr>
<tr>
<td>• highliteObject(), to select an object (regardless of the object type) based on its name.</td>
<td></td>
</tr>
</tbody>
</table>

Data structures involved: **NONE**
Component: **EditorAirspaceRenderer**  
Interface: **SetVisualOptions**

**Description:**
The SetVisualOptions interface allows to setup the graphical settings of the editor main view-port, through the following methods:

- `setGridColor()`, to set the colour of the background grid;
- `setGridSpacing()`, to set the spacing of the background grid;
- `showFixLabels()`, to enable/disable the visualisation of Fix labels;
- `showConflicts()`, to enable/disable the visualisation of conflicts;
- `setCreatingPlan()`, to enable a different visualisation of the mouse cursor while a flight plan is being created;

**Data structures involved:** NONE

### 3.3.3 Data structures
This section describes the main data structures used in the Experimental Testbed.

For each data structure, all its member variables are reported and described in a table. At implementation level, each data structure is implemented by a class in the object-oriented Java language.

**Airspace**

**Description:**
This data structure contains the necessary data to describe the airspace structure, including geographic location and extension and elements contained in it (fixes, airplanes flight-plans, weather objects).

**Member variables:**

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_Fixes</td>
<td>Vector of type AirspaceFix</td>
<td>Vector whose elements represent all the fixes (waypoints, VORs, etc.) contained in the airspace</td>
</tr>
<tr>
<td>m_Clouds</td>
<td>Vector of type Cloud</td>
<td>Vector whose elements represent all the weather objects (cumulonimbi and similar) contained in the airspace</td>
</tr>
<tr>
<td>m_SouthLat</td>
<td>double</td>
<td>Double-precision, floating point value of the latitude of the south boundary of the airspace</td>
</tr>
<tr>
<td>Data Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>m_NorthLat</td>
<td>double</td>
<td>Double-precision, floating point value of the latitude of the north boundary of the geographic region of interest for this airspace</td>
</tr>
<tr>
<td>m_WestLon</td>
<td>double</td>
<td>Double-precision, floating point value of the longitude of the western boundary of the geographic region of interest for this airspace</td>
</tr>
<tr>
<td>m_EastLon</td>
<td>double</td>
<td>Double-precision, floating point value of the longitude of the eastern boundary of the geographic region of interest for this airspace</td>
</tr>
<tr>
<td>m_FlightPlans</td>
<td>Vector of type FlightPlan</td>
<td>Vector whose elements represent all the flight plans crossing the airspace</td>
</tr>
</tbody>
</table>

**AirspaceFix**

**Description:**
This data structure contains information used to describe a so-called “Fix” in the airspace (e.g. waypoints).

**Member variables:**

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_Name</td>
<td>String</td>
<td>Text string containing the name (i.e. the alphanumeric code) of the Fix</td>
</tr>
<tr>
<td>m_Type</td>
<td>String</td>
<td>Text string representing the type of fix (e.g. “Waypoint”, “VOR”, etc.)</td>
</tr>
<tr>
<td>m_Latitude</td>
<td>double</td>
<td>Double-precision, floating point value of the Fix latitude.</td>
</tr>
<tr>
<td>m_Longitude</td>
<td>double</td>
<td>Double-precision, floating point value of the Fix longitude.</td>
</tr>
</tbody>
</table>

**Cloud**

**Description:**
This data structure contains the information describing a weather object in the airspace; this is
meant to include weather objects like cumulonimbi (i.e. significantly dangerous clouds) or other volumes like turbulent regions. However, these objects (generically called “clouds”) have a simplified representation, consisting of their position, altitude extension and horizontal extension.

### Member variables:

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_Name</td>
<td>String</td>
<td>Text string containing the alphanumeric code (name) used by the software application to identify each weather object</td>
</tr>
<tr>
<td>m_Lat</td>
<td>double</td>
<td>Double-precision, floating point value of the centre point latitude</td>
</tr>
<tr>
<td>m_Lon</td>
<td>double</td>
<td>Double-precision, floating point value of the centre point longitude</td>
</tr>
<tr>
<td>m_BaseHgtFeet</td>
<td>double</td>
<td>Double-precision, floating point value of the altitude (in feet) of the bottom level (floor) of the cloud volume</td>
</tr>
<tr>
<td>m_TopHgtFeet</td>
<td>double</td>
<td>Double-precision, floating point value of the altitude (in feet) of the top level (ceiling) of the cloud volume</td>
</tr>
<tr>
<td>m_RadiusKM</td>
<td>double</td>
<td>Double-precision, floating point value of the radial extension (in km) of the cloud volume in the horizontal dimension.</td>
</tr>
</tbody>
</table>

### FlightPlan

**Description:**
This data structure contains the information describing a (simplified) aircraft flight plan.

### Member variables:

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_IDCode</td>
<td>String</td>
<td>Text string containing the alphanumeric code used to identify a flight (usually corresponding to an aircraft call-sign, e.g. “AZ 1234”)</td>
</tr>
<tr>
<td>m_AircraftType</td>
<td>String</td>
<td>Text string describing the type of aircraft (e.g. “B747”)</td>
</tr>
<tr>
<td>m_Description</td>
<td>String</td>
<td>Text string containing a short description of the flight (e.g.</td>
</tr>
</tbody>
</table>
"London to Brussels")

<table>
<thead>
<tr>
<th>Member variable</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_StartTime</td>
<td>ClockTime</td>
<td>An object of type ClockTime, containing the value (hrs, min, sec) of the planned departure time of the flight</td>
</tr>
<tr>
<td>m_DefaultSpeedKts</td>
<td>double</td>
<td>Double precision, floating point value of the cruise speed (in knots)</td>
</tr>
<tr>
<td>m_DefaultFL</td>
<td>double</td>
<td>Double precision, floating point value of the cruise flight level (note: a flight level unit is equivalent to 100 feet, e.g. FL 300 represents 30,000 feet)</td>
</tr>
<tr>
<td>m_WPs</td>
<td>Vector of type FlightPlanWP</td>
<td>Vector whose elements represent the sequence of waypoints used by flight plan to define the path to the final destination</td>
</tr>
</tbody>
</table>

**FlightPlanWP**

**Description:**
This data structure contains the information describing a waypoint used to define a flight-plan; this data structure differs from the “AirspaceFix” data structure in that it contains additional information that are relevant to the flight plan, such as the planned altitude, speed and time at which the aircraft will fly through the waypoint.

**Member variables:**

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_Fix</td>
<td>AirspaceFix</td>
<td>An object of type AirspaceFix, specifying a Fix used as waypoint in a flight plan.</td>
</tr>
<tr>
<td>m_SpeedKts</td>
<td>double</td>
<td>Double-precision, floating point value of the planned speed (in knots) at which the aircraft should fly through the waypoint</td>
</tr>
<tr>
<td>m_FlightLevel</td>
<td>double</td>
<td>Double-precision, floating point value of the planned flight level at which the aircraft should fly through the waypoint</td>
</tr>
<tr>
<td>m_Time</td>
<td>ClockTime</td>
<td>Object of class ClockTime containing the value of the foreseen time at which the aircraft should fly through the waypoint.</td>
</tr>
</tbody>
</table>
SimAircraft

Description:
This data structure contains the information describing a simulated aircraft.

Member variables:

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_ID</td>
<td>String</td>
<td>Text string containing the alphanumeric code used to identify the aircraft</td>
</tr>
<tr>
<td>m_Type</td>
<td>String</td>
<td>Text string describing the type of aircraft (e.g. “B747”)</td>
</tr>
<tr>
<td>m_FPlan</td>
<td>FlightPlan</td>
<td>Pointer to a FlightPlan object (see FlightPlan data structure table for details), i.e. the reference flight plan for the aircraft</td>
</tr>
<tr>
<td>m_Lat</td>
<td>double</td>
<td>Double-precision, floating point value of the aircraft latitude</td>
</tr>
<tr>
<td>m_Lon</td>
<td>double</td>
<td>Double-precision, floating point value of the aircraft longitude</td>
</tr>
<tr>
<td>m_FL</td>
<td>double</td>
<td>Double precision, floating point value of the aircraft flight level</td>
</tr>
<tr>
<td>m_SpeedKts</td>
<td>double</td>
<td>Double precision, floating point value of the aircraft speed (in knots).</td>
</tr>
<tr>
<td>m_VSpeedFeetSec</td>
<td>double</td>
<td>Double precision, floating point value of the aircraft vertical speed (in feet per second).</td>
</tr>
<tr>
<td>m_HeadingDeg</td>
<td>double</td>
<td>Double precision, floating point value of the aircraft heading (in degrees with respect to the North).</td>
</tr>
<tr>
<td>m_Finished</td>
<td>boolean</td>
<td>Boolean value (true or false) telling whether an aircraft has completed its flight plan.</td>
</tr>
</tbody>
</table>

Scenario

Description:
This data structure contains the information used to describe a (simplified) air traffic scenario. The data include: an airspace description, which comprises flight plans (aircraft), weather objects and fixes, plus complementary information like a textual description of the scenario, a rating of the level of difficulty of the scenario and a list of potential events such as conflicts between aircraft and
crossing of weather objects.

### Member variables:

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_Airspace</td>
<td>Airspace</td>
<td>An object of type Airspace, specifying the reference airspace for this scenario</td>
</tr>
<tr>
<td>m_Description</td>
<td>String</td>
<td>A text string containing the narrative description of the scenario or any other user-defined textual comments on the scenario</td>
</tr>
<tr>
<td>m_Difficulty</td>
<td>int (integer)</td>
<td>Integer value of the difficulty score associated to the scenario</td>
</tr>
<tr>
<td>m_Conflicts</td>
<td>Vector of type Conflict</td>
<td>Vector whose elements represent the set of (potential) aircraft-aircraft conflict events detected in the scenario</td>
</tr>
<tr>
<td>m_AirCloudIns</td>
<td>Vector of type Conflict</td>
<td>Vector whose elements represent the set of (potential) cloud-aircraft intersection events</td>
</tr>
</tbody>
</table>

### Conflict

**Description:**

This data structure contains the information used to describe either a conflict event between two aircraft or an intersection of an aircraft with a weather object (cloud).

### Member variables:

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_Time</td>
<td>ClockTime</td>
<td>An object of class ClockTime containing the value of the time (hrs, min, sec) at which the event occurs</td>
</tr>
<tr>
<td>m_Distance</td>
<td>double</td>
<td>Double-precision, floating point value of the distance between the two conflicting entities (aircraft-aircraft or aircraft-cloud) at the time of the event occurrence</td>
</tr>
<tr>
<td>m_Latitude</td>
<td>double</td>
<td>Double-precision, floating point value of the latitude of the conflict position (conflict position is conventionally assumed to be at the mid-point</td>
</tr>
</tbody>
</table>
between the two conflicting entities)

| m_Longitude  | double          | Double-precision, floating point value of the longitude of the conflict position (conflict position is conventionally assumed to be at the mid-point between the two conflicting entities) |

**AircraftConflict**

**Description:**
This data structure derives from the *Conflict* one, specialising the scope to the only case of conflict between two airplanes. In addition to the data contained in Conflict, this data structure includes the identification codes of the two conflicting airplanes.

**Member variables:**

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_Aircraft1_ID</td>
<td>String</td>
<td>Text string containing the identification code of one of the two conflicting airplanes</td>
</tr>
<tr>
<td>m_Aircraft2_ID</td>
<td>String</td>
<td>Text string containing the identification code of one of the two conflicting airplanes</td>
</tr>
</tbody>
</table>

**AircraftCloudConflict**

**Description:**
This data structure derives from the *Conflict* one, specialising the scope to the only case of intersection of a cloud by an aircraft. In addition to the data contained in Conflict, this data structure includes the identification codes of the aircraft and the cloud.

**Member variables:**

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_Aircraft_ID</td>
<td>String</td>
<td>Text string containing the identification code of the intersecting aircraft</td>
</tr>
<tr>
<td>m_Cloud_ID</td>
<td>String</td>
<td>Text string containing the identification code of the cloud crossed by the aircraft</td>
</tr>
</tbody>
</table>
3.4 Interfaces to external components

This section describes the interface of the Experimental Testbed to the external components, which in this case consist in the 3D/2D HMI prototypes.

This part only applies to the scenario simulation module, not to the scenario editor (which constitutes a separate module).

The interface used is based on a Client-Server communication architecture.

The simulator (testbed) plays the role of the Server, while the HMI prototypes are the clients connecting to the server.

The underlying communication protocol used is the well-known TCP-IP (Transmission Control Protocol – Internet Protocol).

3.4.1 TCP protocol

TCP provides a point-to-point channel for applications that require reliable communications.

When two applications want to communicate to each other reliably, they establish a connection and send data back and forth over that connection.

If the client application wants to send/receive data to/from a server, the client needs to create a socket (client socket) and use this socket to communicate with the server. At the server side, there are two kinds of sockets. One is the listening socket, which is responsible to take care of requests from new clients. The other is the service socket whose job is to take care of all the communication between existing clients and the server. Once a new user requests to join the service, the listening socket will create a new service socket for this new user. The functionality of the service socket is like that of a representative of each client. In other words, if there are N users in this system, there will be N service sockets in the server side.

The following figure illustrates the client-server communication scheme using sockets.
3.4.2 Client - Server communication

This paragraph describes the specific mechanism implemented for exchanging data between the server (simulator) and the client.

This mechanism can be described as follows:

- the server is started and put in ‘listening mode’; in such a mode, the server waits for a connection attempt from a TCP client;
- the client tries to connect to the server, using the IP address of the server and the port number on which the server is listening (if the client and the server are running on the same machine, the “127.0.0.1” address shall be used);
- the server accepts and establishes the TCP connection with the client;
- once the connection is established, the server periodically (e.g. once every 10 milliseconds) checks for the presence of messages coming from the client through the TCP, and reads them;
- the client periodically sends a message to the server; the message can be of one of the following types:
  - request: this kind of message is sent to the server in order to receive back updated data from the server itself, (i.e. from the simulator); for instance, the client may send a request asking to receive the updated position of all aircraft;
  - event notification: this kind of message is sent to the server to notify it that the a client-specific event has occurred; for instance a client may send an event
notification to the server to inform that in the client HMI the user has performed some actions (e.g. the user has selected an aircraft, etc.); the event notification can in practice contain any text message. Notification of time-tagged events related to the HMI (client) actions can be considered a useful means for supporting evaluation of the HMI by analyzing the user inputs and compare them with traffic conditions.

- Whenever the server receives a message from the client, it first identifies the type of message, in order to discriminate between ‘requests’ and ‘event notifications’;
- if the received message is a request, the server performs the following operations:
  o it processes the message in order to identify the specific request;
  o it prepares a data package according to the request (e.g. if the client is requesting the data of an aircraft, the server will prepare a data package containing those information);
  o it sends the data package back to the client;
- if the received message is an event notification, the server performs the following operations:
  o it determines the time (simulation time) when the message was received;
  o it reads the message content;
  o it records the time and message content into an event log file (".data/eventlog.txt");
- at the same time, the client periodically checks for new available data coming from the server, and reads them.

In order to allow the server to discriminate between an event notification message and a request message, the client has to add a prefix ahead of each message:
- REQ_: this prefix allows the server to identify the message as a data request;
- EVENT_: this prefix allows the server to identify the message as a client event notification.

**Processing client requests**

As outlined above, whenever the server receives a request message from the client, the first action taken by the server is to process the request so to identify it.

In fact, each request message will consist of a text string containing a keyword, which should belong to a predefined list of keywords that the server can recognize. If the keyword does not match any of the predefined ones, the request is not recognized and thus ignored.

The request keywords recognized by the server are listed here:

- **SIM_DATA**: by using this keyword, the client asks the server for general simulation data, such as the current simulation time;
- **ALL_AC_DATA**: by using this keyword, the client asks the server for a data package containing updated information on all the airplanes that are currently flying in the airspace (position, flight level, speed, heading, id-code, etc.);
- **ALL_FP_DATA**: by using this keyword, the client asks the server for a data package containing updated information on all the flight plans in the airspace;
- **ALL_FIXES_DATA**: by using this keyword, the client asks the server for a data package containing the information (position, name, type) of all the fixes in the airspace;

- **ALL_CLOUDS_DATA**: by using this keyword, the client asks the server for a data package containing the data (position, name, extension) of all the clouds (i.e. cumulonimbi and similar) in the airspace.

- **N_ITEMS**: by using this keyword, the client asks the server for the total number of objects in the airspace, i.e. the number of aircraft, the number of fixes and the number of weather objects.

### Sending data packages to the client

Once the request has been identified, a set of data is prepared accordingly and sent back to the client, in response to the specific request.

For instance, if the client is requesting the data of an aircraft (position, speed, flight level, etc.), a package containing the aircraft data is prepared, using the latest available data in the simulator, and then sent to the requesting client through the TCP connection.

The response data package is built by the server application according to the following rules:

- each data package begins with a conventional label identifying the type of data; for instance, the “AC_DATA” label identifies a data package containing all the relevant information of an aircraft, i.e. its position, flight level, speed, heading, id-code (call-sign) and type of aircraft;

- the data package label is followed by a list of data fields; an example of data field could be the speed of an aircraft, or its flight level, etc.;

- each data field consists of two parts, i.e. a label followed by the specific value of the data; for instance, the data field “AC_SPEED 350” would mean: “aircraft speed = 350 knots”.

Obviously, the client application receiving the data package will have to apply the same rules in order to extract the information from such the message representing such a data package.

The list of data package and data field labels used by the server and client applications are listed here below:

<table>
<thead>
<tr>
<th>Label</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC_DATA</td>
<td>Data package</td>
<td>Data package containing an aircraft information (id, type, speed, flight level, position, heading)</td>
</tr>
<tr>
<td>AC_INDEX</td>
<td>Data field</td>
<td>Aircraft index (0,1,2,...) in the list of all aircraft in the airspace</td>
</tr>
<tr>
<td>AC_ID</td>
<td>Data field</td>
<td>Aircraft identification code (call-sign)</td>
</tr>
<tr>
<td>AC_TP</td>
<td>Data field</td>
<td>Aircraft type (e.g. “B747”)</td>
</tr>
<tr>
<td>AC_SPD</td>
<td>Data field</td>
<td>Aircraft speed</td>
</tr>
<tr>
<td>Data Field</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>AC_LAT</td>
<td>Aircraft latitude (position)</td>
<td></td>
</tr>
<tr>
<td>AC_LON</td>
<td>Aircraft longitude (position)</td>
<td></td>
</tr>
<tr>
<td>AC_HDG</td>
<td>Aircraft heading</td>
<td></td>
</tr>
<tr>
<td>AC_FL</td>
<td>Aircraft flight level</td>
<td></td>
</tr>
<tr>
<td>AC_VSPD</td>
<td>Aircraft vertical speed (rate of climb)</td>
<td></td>
</tr>
<tr>
<td>CLOUD_DATA</td>
<td>Data package containing a weather object information</td>
<td></td>
</tr>
<tr>
<td>CLOUD_INDEX</td>
<td>Cloud index (0,1,2,…) in the list of all clouds in the airspace</td>
<td></td>
</tr>
<tr>
<td>CLOUD_NAME</td>
<td>Cloud (weather object) identification code</td>
<td></td>
</tr>
<tr>
<td>CLOUD_LAT</td>
<td>Cloud latitude</td>
<td></td>
</tr>
<tr>
<td>CLOUD_LON</td>
<td>Cloud longitude</td>
<td></td>
</tr>
<tr>
<td>CLOUD_BASE_HGT</td>
<td>Cloud base (floor) altitude</td>
<td></td>
</tr>
<tr>
<td>CLOUD_TOP_HGT</td>
<td>Cloud top (ceiling) altitude</td>
<td></td>
</tr>
<tr>
<td>FIX_DATA</td>
<td>Data package containing a Fix information</td>
<td></td>
</tr>
<tr>
<td>FIX_INDEX</td>
<td>Fix index (0,1,2,…) in the list of all the fixes in the airspace</td>
<td></td>
</tr>
<tr>
<td>FIX_NAME</td>
<td>Fix identification code</td>
<td></td>
</tr>
<tr>
<td>FIX_LAT</td>
<td>Fix latitude</td>
<td></td>
</tr>
<tr>
<td>FIX_LON</td>
<td>Fix longitude</td>
<td></td>
</tr>
<tr>
<td>FIX_TYPE</td>
<td>Type of Fix</td>
<td></td>
</tr>
<tr>
<td>FP_DATA</td>
<td>Data package containing a flight plan information</td>
<td></td>
</tr>
<tr>
<td>FP_INDEX</td>
<td>Index of the flight plan (0,1,2,…) in the list of all flight plans</td>
<td></td>
</tr>
<tr>
<td>FP_ID</td>
<td>Flight plan identification code</td>
<td></td>
</tr>
<tr>
<td>FP_WP</td>
<td>Name (id code) of a waypoint in the flight plan</td>
<td></td>
</tr>
<tr>
<td>SIM_DATA</td>
<td>Data package containing general data on the state of the simulation.</td>
<td></td>
</tr>
<tr>
<td>SIM_TIME</td>
<td>Simulation time</td>
<td></td>
</tr>
<tr>
<td>N_ITEMS</td>
<td>Data package containing information on the number of objects (aircraft, fixes, clouds, etc.)</td>
<td></td>
</tr>
<tr>
<td>N_ACS</td>
<td>Data field</td>
<td>Number of aircraft in the scenario</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>N_FIXES</td>
<td>Data field</td>
<td>Number of fixes in the scenario</td>
</tr>
<tr>
<td>N_CLOUDS</td>
<td>Data field</td>
<td>Number of weather objects in the scenario</td>
</tr>
</tbody>
</table>
4. PART 3: USER INTERFACE

This section describes the user interface of the Experimental Testbed, covering both the scenario editor and the scenario player (simulator).

It provides an overview of the graphical interface structure, describing the graphical widgets used and how to use them for accessing the software capabilities.

4.1 Scenario Editor

This section describes the scenario editor user interface.

The editor is presented to the user through a window providing a graphical user interface.

The editor window is divided into 5 main areas:

- a 2D airspace view-port, located in the centre of the editor window;
- a menu bar, located at the top of the editor window;
- a control bar, located at the bottom of the editor window;
- a control panel, located on the left of the editor window;
- a tree-view of the airspace elements, located on the right of the editor window;

![Figure 6. Scenario Editor: overview](image-url)
4.1.1 Airspace 2D view

The Airspace 2D appears at the centre of the Editor window. It is used to represent the basic airspace structure and its relevant element, so that the user can have an immediate visual feedback on the scenario that s/he is creating.

Besides, some of the editing operations (e.g. Flight Plan creation) can be easily performed by directly selecting the graphical elements displayed on the 2D airspace view.

Figure below provides an example of airspace displayed in the Editor.

![Figure 7. Scenario Editor: airspace 2D view](image)

The Airspace 2D view can represent the following elements:

- the geographic area of interest, represented by a “latitude-longitude” grid over a black background;
- Fixes, represented as little white triangles, with or without labels (label visualization can be enabled or disabled by the user);
- flight plans, as dotted lines connecting a list of waypoints (fixes);
- aircraft, represented as green dots, each one accompanied by a label with indication of the ID code (call-sign) and the flight level;
- weather objects (“clouds”), represented by transparent circles whose radius corresponds to the horizontal (2D) extension of these objects;
- conflicts between airplanes and intersections between an aircraft and a weather object, represented by a red/orange circle.
Figure 8. Scenario Editor: conflict / intersections visualisation

4.1.2 Editor Menu Bar
The Editor Menu Bar is the part of the GUI located at the top of the Editor window. It includes a pair of menus that can be accessed by the user by clicking on their labels. These menus are:

- the File menu;
- the Scenario Menu.

The File menu (figure below) provides access to basic functions for accessing the file system to load and save airspaces and scenarios (stored as XML files) created with the Editor.

Figure 9. Scenario Editor: File menu

The Scenario Menu provides access to a number of editing functions:

- **Add New Fix**, to create a new Fix in the airspace;
- **Add Random Fixes**, to ask the system to automatically create a specified number of fixes in the airspace (randomly generated).
- **Add New Flight Plan**, to create a new flight plan;
- **Add New Cloud**, to create a new cloud (weather object);
- **Airspace Properties**, to edit the airspace settings, such as the geographic extension;
- **Edit Scenario**, to edit generic scenario characteristics, including level of difficulty and textual comments or description;
- **Find Conflicts**, to ask the system to automatically detect any potential conflicts in the scenario (between two aircraft) or aircraft-cloud intersections.

![Figure 10. Scenario Editor : Scenario menu](image)

### 4.1.3 Editor Control Bar

The Editor Control Bar is the part of the GUI located at the bottom of the Editor window. It includes a slider that allows the user to interactively control the temporal dimension of the scenario.

Specifically, by moving the slider back and forth, the user can set the value of the time (expressed in minutes and seconds) along the timeline of the scenario which is being created and see the corresponding position changes of the aircraft along their flight plans.

![Figure 11. Scenario Editor: time control slider](image)

![Figure 12. Scenario Editor: aircraft move as the time slider is shifted](image)
4.1.4 Editor Control Panel

The Editor Control Panel is the part of the GUI located on the left side of the Editor window. It includes several elements such as buttons, checkboxes and spinners which are used to access some of the main functions (as an alternative to the menu bar) as well as to control the display settings of the editor.

This panel also displays the geographic boundaries of the current region (South and North latitude, West and East longitude).

![Figure 13. Scenario Editor – Control panel](image)

The functions accessible through this panel are:

- **New Flight Plan**: allows the user to create a new flight plan;
- **Remove Flight Plan**: allows the user to remove a selected flight plan from the airspace;
- **Edit Flight Plan**: allows the user to modify a selected flight plan;
- **New Fix**: allows the user to create a new Fix in the airspace;
- **New Cloud**: allows the user to create a new cloud (weather object);
- **Random Fixes**: asks the system to create a set of randomly located fixes;
- **Find Conflicts**: asks the system to detect any potential conflict (aircraft-aircraft) and aircraft-cloud intersections;
- **Show Fix Labels**: enable/disable the visualization of labels of Fixes in the 2D airspace view;
- **Show Conflicts**: enable/disable the visualization of conflicts detected by the system.
- **Grid Spacing**: allows the user to specify the grid spacing, i.e. the distance (Km) between two horizontal and vertical lines in the grid displayed in the 2D airspace view.
4.1.5 Editor Tree-Viewer

This GUI element is located on the right side of the Editor window and provides a hierarchical view of the elements which have been inserted in the airspace. This representation is based on the use of a tree-like graphical structure, with a root element (the airspace) and a number of branches and sub-branches ending with leaf elements.

![Figure 14. Scenario Editor: airspace tree-viewer](image)

The root element is the airspace under creation. Children under the root are grouped in the following categories (branches):

- Fixes;
- Flight Plans;
- Clouds

Under the Fixes branch, the list of all the Fixes (leaf nodes) in the airspace in presented. Under the Clouds branch, the list of all the weather objects in the airspace is presented; each Cloud element can be exploded, showing its relevant data, such as position, extension, etc.

Under the Flight Plans branch, the list of all the flight plans is presented; each flight plan element can be exploded into its children elements, which are the flight plan data (id-code, speed, start time, etc.) and the list of waypoints crossed by the plan itself.

The tree can be exploded or collapsed at the desired level of detail through a simple mouse click on the branches; elements in the tree can be selected with a simple mouse-click on the node label; in particular, by selecting a flight plan or a Fix node, the corresponding elements will be highlighted in the 2D airspace view.
4.1.6 Creating a Flight Plan

This chapter goes through a description of a typical flight plan creation process, assuming that the user is starting from a blank (i.e. empty) scenario.

Step 1: Creating Fixes

The first step is to populate the region of interest with a certain number of Fixes, which will then be used as waypoints for creating the actual flight plans.

A first approach consists in entering the Fixes one by one, manually specifying their geographic coordinates, their name (code) and type (figure below).

![Figure 15. Scenario Editor: Dialog for adding a Fix](image)

A more rapid approach consists in asking the system to automatically generate a given number of Fixes (to be specified by the user). In this way, the airspace can be quickly populated with a big number of waypoints that can then be used to create the flight paths.

![Figure 16. Scenario Editor: adding random Fixes](image)

Step 2: Creating a Flight Plan

Once the airspace has been populated by Fixes, these ones can be used as waypoints to build a flight path through the airspace.

The user will first specify the general settings of the new flight plan, such as the ID code (call sign), the aircraft type, the start time, the cruise speed and altitude, and (optionally) a short description of the flight plan (e.g. "Route from London to Brussels").
The user can also specify whether s/he wants to use a constant value for the speed and the flight level all along the flight path, or to set a different value at each waypoint; obviously the first option (using constant value) allows for a much quicker creation of the flight plan, as the user will not be asked to enter speed and flight level at each waypoint, while the second option allows the user to define the flight path in a more precise and realistic way.

**Figure 17. Scenario Editor: setting up a flight plan**

Once the general settings have been inputted, the user is asked to sketch a poly-line through a number of waypoints of his/her choice, by sequentially clicking on the waypoints represented in the 2D airspace view, as shown in figure below. If the constant value (speed and altitude) option is not used, a pop-up dialog will be displayed for each clicked point asking the user to enter the local speed and flight level values.

**Figure 18. Scenario Editor: creating a flight plan**
4.1.7 Adding Weather Objects
As said in previous sections, it is possible to add a weather object to the airspace. For reasons of simplicity, weather objects are also referred to as “clouds” in the editor; in fact, they represent 3D volumes within the airspace, characterized by a position (lat, long), a horizontal extension (radial extension) and a vertical extension. These volumes can represent a region of high turbulence, a cumulonimbus (or a region containing cumulonimbi) or other weather phenomena which can be though of as bound by a 3D volume in space.

By selecting the Add New Cloud function, either in the Scenario menu or in the Control Panel, a dialog box is displayed asking the user to enter the relevant data for the weather object:

- Cloud name (this is used to identify the object in the scenario);
- Latitude of the centre point of the weather object;
- Longitude of the centre point of the weather object;
- Radial (horizontal) extension of the weather object volume;
- Base altitude of the weather object volume;
- Top altitude of the weather object volume;

![Figure 19. Scenario Editor: adding a new weather object](image)

4.1.8 Adding Scenario Complementary Information
As said in previous sections, the user can add some complementary information to a scenario; this information include a textual description of the scenario itself, which may include any notes or comments, and a score associated to the level of difficulty of the scenario. This information are however completely defined by the user and as such they are based on a subjective assessment of the scenario; however, such information can be useful to quickly recall the salient aspects at a later stage, for instance at the time of experiment execution. This information are saved together with the airspace data, when the user saves the scenario.
4.2 Scenario Player

This section describes the scenario player (simulator) user interface. The simulator is presented to the user through a window providing a graphical user interface. The simulator window is divided into 3 main areas:

- a 2D airspace view-port, located in the centre of the simulator window;
- a menu bar, located at the top of the simulator window;
- a control bar, located at the bottom of the simulator window;

Figure below provides a global view of the simulator window.
4.2.1 Airspace 2D view

The Airspace 2D appears at the centre of the Simulator window. It is used to represent the basic airspace structure and its relevant element, so that the user can have an immediate visual feedback on the scenario that is being simulated. Figure below provides an example of airspace displayed in this view-port.
4.2.2 Simulator Menu Bar

The Simulator Menu Bar is the part of the GUI located at the top of the Simulator window. It includes a pair of menus that can be accessed by the user by clicking on their labels. These menus are:

- the File menu;
- the Simulation menu.

The File menu (figure below) provides access to basic functions for accessing the file system to load airspaces and scenarios (stored as XML files) previously created with the Editor.

The Simulation menu provides access to a group of functions for controlling the run of the simulation and for getting information on the simulated scenario:

- **Start Sim**, to start the running of the simulated scenario;
- **Pause Sim**, to pause the simulation execution;
- **Resume Sim**, to resume the simulation execution after a pause;
- **Stop Sim**, to stop the simulation execution;
- **Show Scenario Info**, to display the complementary text info associated to the current scenario.

![Figure 24. Simulator: simulation menu](image)

### 4.2.3 Simulator Control Bar

The Simulator Control Bar is the part of the GUI located at the bottom of the Simulator window. It includes the following graphical controls:

- a checkbox to enable/disable the visualisation of Labels of Fixes in the airspace;
- a grid spacing spinner, to set the distance between parallel lines in the grid displayed in the background of the airspace;
- a time field, showing the current time (hours, minutes and seconds) in the simulated scenario;
- a slider to control the speed of execution of the simulation, where “1 X” means that the simulation runs in real time, while values greater than 1 mean that the simulation runs at an accelerated speed with respect to real time execution.

![Figure 25. Simulator: control bar](image)
APPENDIX A – SOFTWARE CODE DOCUMENTATION

This appendix contains the software code documentation.

The code has entirely been written in the Java language, which ensures cross-platform portability and Object-Oriented design.
END OF DOCUMENT