D4.1 Description of the validation platform

Deliverable ID:	**D4.1**
Project acronym:	**ASTAIR**
Grant:	**101114684**
Call:	**HORIZON-SESAR-2022-DES-ER-01**
Topic:	**HORIZON-SESAR-2022-DES-ER-01-WA1-1**
Consortium coordinator:	**Ecole Nationale de L'Aviation Civile (ENAC)**
Edition date:	**13 February 2025**
Edition:	**01.00**
Status:	**Official**
Classification:	**PU**

Abstract

This deliverable presents the results of user studies with ground controllers and ground handlers as well as workshops with the consortium members and the stakeholder consultation group for identifying and narrowing the first set of expectations of the project. In particular we identified use cases with several automation levels that will be explored in WP2 (Support algorithms) and WP3 (Automation Supervision & Control HMI design and development.

Authoring & approval

Author(s) of the document

Organisation name	Date

ENAC	27/01/2025

Reviewed by

Organisation name	Date
TU Delft	12/2/2025
ENAC	12/2/2025
Deep Blue	12/2/2025
ADP	12/2/2025
Eurocontrol	12/2/2025

Approved for submission to the SESAR 3 JU by[1]

Organisation name	Date
TU Delft	12/2/2025
ENAC	12/2/2025
Deep Blue	12/2/2025
ADP	12/2/2025
Eurocontrol	12/2/2025

Rejected by[2]

Organisation name	Date

Document history

Edition	Date	Status	Company Author	Justification
00.01	27/01/2025	Draft	ENAC	Initial version
01.00	13/02/2025	Release	ENAC	Correction after internal review

[1] Representatives of all the beneficiaries involved in the project

[2] Representatives of the beneficiaries involved in the project

ASTAIR

AUTO-STEER TAXI AT AIRPORT

ASTAIR

This document is part of a project that has received funding from the SESAR 3 Joint Undertaking under grant agreement No 101114684 under European Union's Horizon Europe research and innovation programme.

Table of contents

Co-funded by the European Union

List of figures

Executive Summary

The goal of the ASTAIR project is to design a seamless partnership between Human and Artificial Intelligence (AI) to manage and perform engine-off and conventional airport surface movement operations at major European airports. ASTAIR original approach to automation is to consider an integrated airport system instead of many separate sub-systems, analyse the level of autonomy an AI system could take on tasks and to make the automation controllable by humans at different levels.

With the introduction of high-level automation for airport surface movement operations, the role of operators and airport operation procedures will significantly change. The key to optimize the overall performance of the collaboration between humans and AI is to adapt intelligent systems to the operators' modus operandi. This will ensure logical consistency across manual and automated control and reduce the cognitive distance between levels of automation by mapping system functions to goals and mental model of operators. In ASTAIR, we will propose interactive tools and adaptative AI algorithms that take advantage of operators' expertise for controlling and engaging with the automation at diverse levels.

This document presents the software architecture developed to demonstrate the algorithms and human-machine interfaces developed during the project on two target airports and following selected uses cases. To cope with integration issues on this low TRL (Technology Readiness Level) project between AI and interfaces, a chain of tools has been developed to forge data simulating the desired use cases. Finally, the simulation setup consists in a small number of software agents each simulating an operational system and interconnected via a simple middleware.

The platform allows to simulate a complete ATC Ground working position, with a radar image, out of window view and other services. The main components are a Radar engine that simulates detected vehicles positions, an A-SMGCS prototype that is enriched for the need of the project, an AI agent that computes the conflict free routing, a scenario orchestrator that animates the platform to create specific situations.

The document presents the setup for AI real time validations. It also explains how the simulation data are generated from a variety of data sources and the two modes of simulation to emphasize either the HMI interactions or the AI computations' validation.

1 Introduction

1.1 Purpose of this document

This document describes the software architecture that allows the integration of the different modules implemented for the project. In addition, the process to create the simulation data used to generate the validation use cases is also presented here.

1.2 Structure of the document

Chapter 2 gives the context of the simulation, the available means and the target airports.

Chapter 3 explains the different data sources used and how they are combined to produce the validation use cases.

Chapter 4 describes the software architecture and data exchanges.

Chapter 5 lists the logging facilities that will help the assessment of the validation sessions.

2 Simulation setup

2.1 Platform and facilities

The ACHIL platform gathers in one place simulators of many working positions of the ATM world: ATC en route, approach and tower positions, cockpit simulators and supervision room. The proximity of air and ground positions makes it possible to focus on air-ground collaboration. The simulation tools employed allow the setup of a very realistic environment for operational experts (AMAN, TCAS, Safety Nets, Aircraft models etc.). Based upon an ad-hoc middleware, the flexibility makes it also compatible with the needs of research and rapid prototyping and offers the possibility to quickly set up a comprehensive operational environment to explore new ideas and concepts.

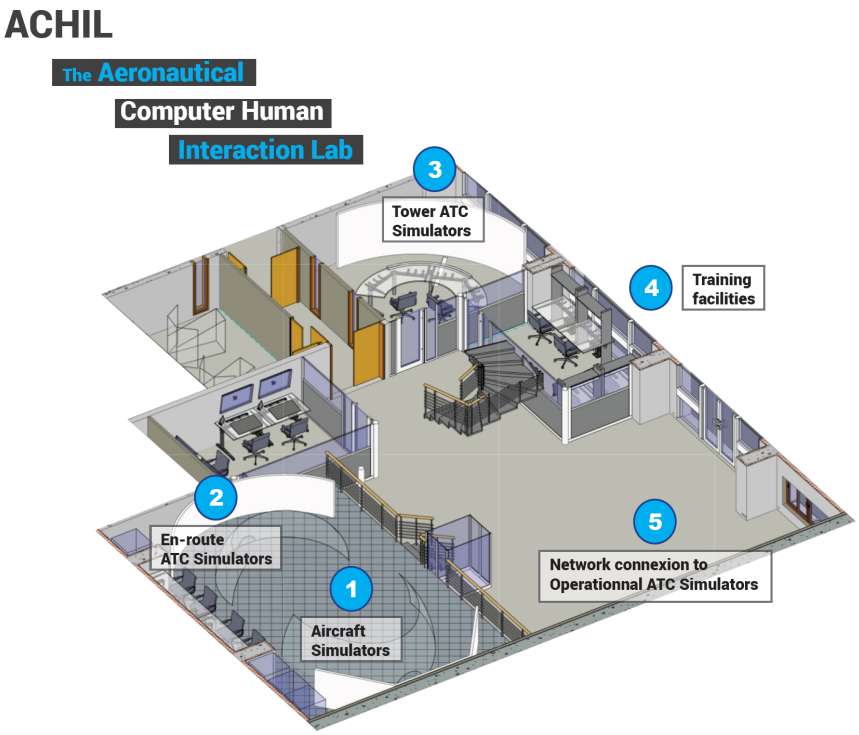

Figure 1: ACHIL simulation facilities

In the frame of ASTAIR project, Ground tower ATC position will be simulated in room 3, see Figure 1.

Figure 2: Ground tower position with RealTwr view

The tower position uses RealTwr[1] for the out of the window as shown in Figure 2. In addition, two tactile displays are available to try different designs for the A-SMGCS and ground supervision interfaces.

In addition, the platform provides means to record the video of the exercises, both screens and users, so that experimentaters can observe the exercises without disturbing the users and post process the videos.

2.2 Target airports and simulation differences

ASTAIR validations shall demonstrate the solution on both Paris Roissy-CDG (LFPG) and Amsterdam Schiphol (EHAM) airports. The validation platform can reproduce both airports' situations, however the different software modules have been developed primarily on one of the two airports. Basically, the multi agents routing system (MAS) is initially tested on EHAM and the supervision HMIs on LFPG. Although the plan is to demonstrate the solution on both airports, to ensure the feasibility of the experiments the MAS will be demonstrated mostly on EHAM traffic, and the user interactions will be tested also with a simulated IA. This latter solution shall ensure that the use cases situations are easily reproductible to focus on interaction design validation.

The normal mode of simulation is with Multi Agent System integrated. In that case the simulation engine provides flight plans, i.e. aircraft type and schedule information and MAS computes all vehicles trajectories in advance, deconflicted with speed regulations, using map and traffic rules data (§3.2). Then it can provide these data to the supervision HMI, so that the human operator can explore the solution by viewing future positions of vehicles. And as the simulation runs, MAS directly sends vehicles positions every second as a radar would do. This setup supposes that the vehicles are following perfectly the computed solution, which is not realistic but sufficient for TRL1 project demonstration.

The simulated AI mode uses a standard A*[7] routing algorithm with taxiways weighted according to airport traffic rules (§3.2.2), but not deconflicted. Then an operator in charge of the data preparation plays the simulation to manually detect conflicts and set speed regulations. These speed regulations

are converted in pilot instructions (§4.3.3.5) that will be send automatically when the simulation is played for validation.

In both cases, the scenario file that describes the vehicles to be generated is described in §3.3.

3 Data sources

3.1 Scenario Definition

The creation of the scenario is based on a flight schedule of arriving and parking flights. The simulation engine can receive messages to generate the corresponding data in the radar tracks database.

For an inbound flight, the simulation engine will compute the radar tracks from 10 miles out in air until the aircraft has vacated the runway. Then, the simulation is either connected to the Multi Agent System and it updates the vehicles positions itself, or the simulation is in simulated AI mode and pilot orders are used to manipulate the vehicles, see §4.3.3.5.

Concerning a departing flight, the simulation engine will generate radar tracks at the parking position for 1 hour. Then the vehicle is either managed by MAS or by pilot orders.

3.2 Map and routing data

ASTAIR project is relying on tools and data developed during former projects. The following sub chapters are describing these parts that ASTAIR is reusing.

3.2.1 Routing information

The initial source for airport maps, i.e. background image but also routing network with all taxi lanes, taxiways and runways defined together with their connections and some geographical information, comes from open source data for the X-Plane flight simulator [3].The description of ground facilities is available for download [4] and the format is proprietary [5]. AEON project team developed a Python program to extract information from these files and generate an SVG file with all the requested information. External data sources such as Open Topo Data [6] have been used to get altitudes of intersections and then compute taxiways slopes.

The routing information inside the SVG file is following the structure below:

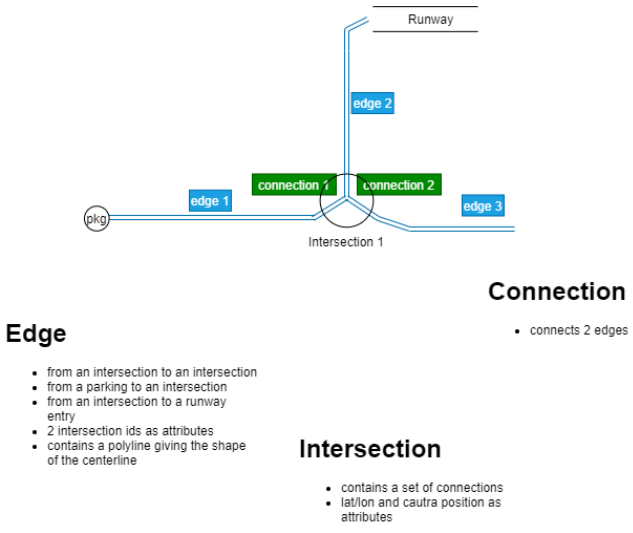

Figure 3: SVG routing data model

An edge defines a portion of taxiway (or taxi lane or runway) between 2 intersections and the intersection node gives the information of possible connection between edges. As the shown in the example below, the edge node is an SVG path element with additional information, including its length, slope and turn radius:

```
<path bound1="intersection_node_1208" bound2="intersection_node_1205"
d="M 171.7197,302.8247 L 171.7199,302.8247" id="edge_1"
pathLength="0.0012" radius="50" slope="0.0"/>
```

And the intersection node gives the connection information:

```
<g id="intersection_node_1205" latitude="49.0129" longitude="2.5058"
x="171.7199" y="302.8247">

    <g bound1="edge_1" bound2="edge_480" id="connection_2505"/>

    <g bound1="edge_480" bound2="edge_800" id="connection_2506"/>

</g>
```

The complete structure of the SVG file is the following:

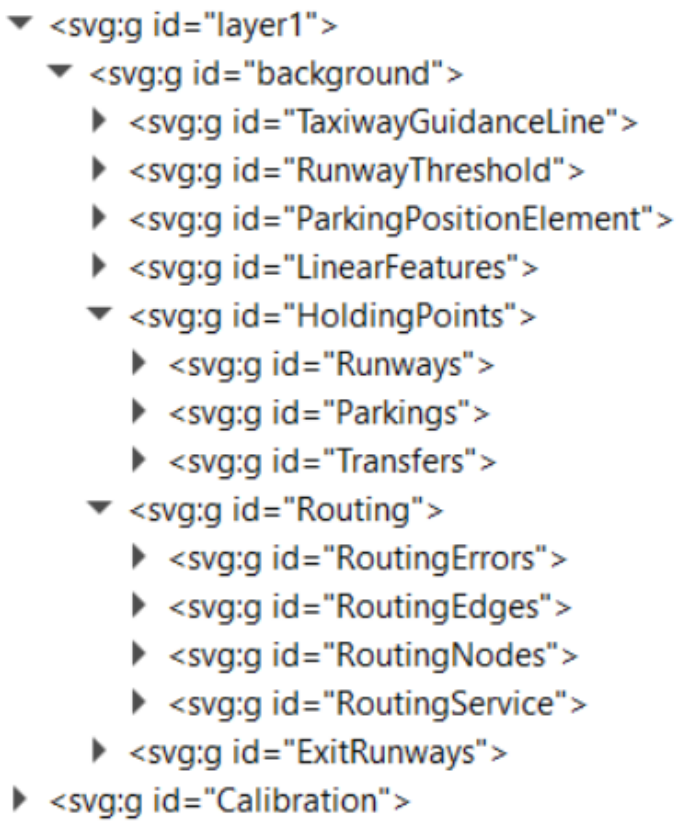

Figure 4: SVG file structure

The first four layers, ‘TaxiwayGuidanceLine’ ‘RunwayThreshold’ ‘ParkingPositionElement’ and ‘Linear Features’, define the background image whereas the ‘HoldingPoints’ ‘Routing’ and ‘ExitRunways’ layers give the routing information that can be processed by routing algorithms. The last layer, ‘Calibration’, gives transformation matrices to switch from latitude/longitude coordinates (or Cautra) to SVG coordinates.

The two figures below represent the maps generated for Amsterdam Schiphol and Roissy CDG airports. The SVG can be graphically represented as this but it also contains all the routing information described above.

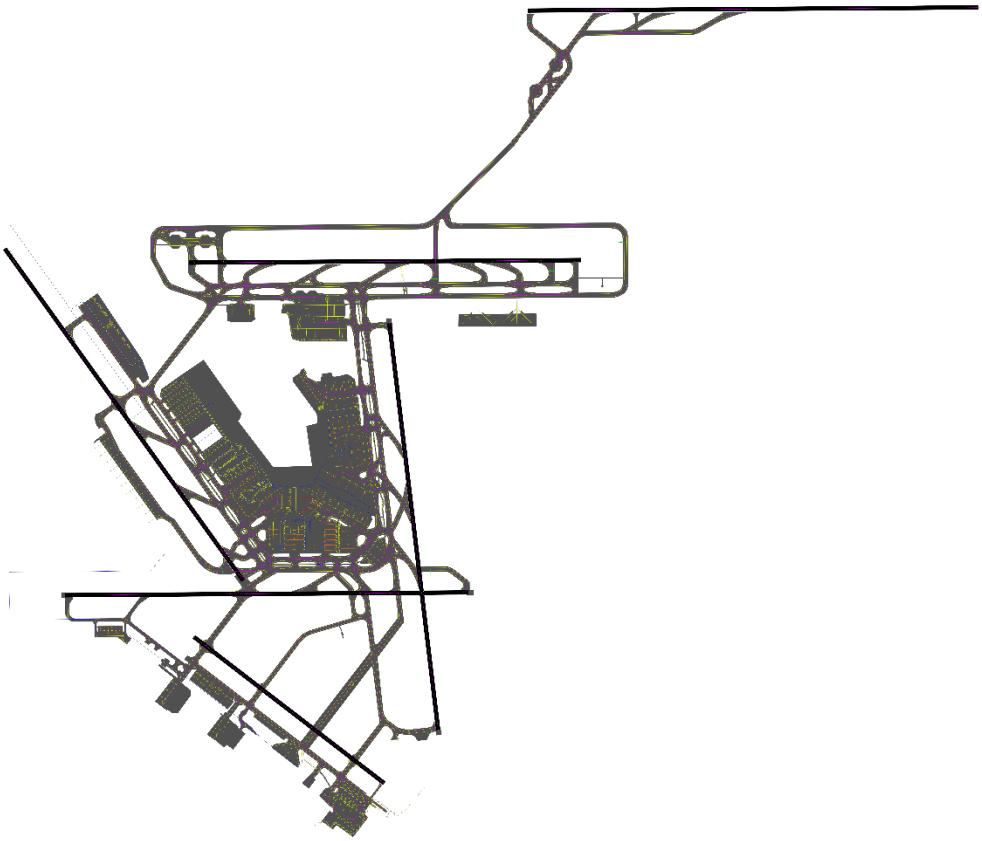

Figure 5: Schiphol (EHAM) SVG map

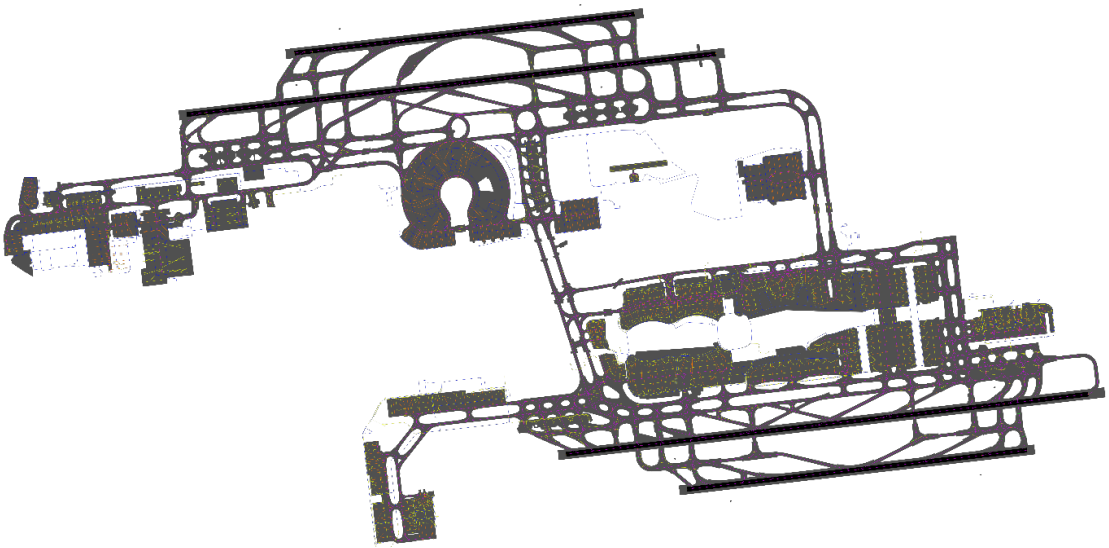

Figure 6: Roissy CDG (LFPG) SVG map

3.2.2 Traffic rules

In order for the routing algorithms to work properly and propose solution that are in line with the operational practices, a definition of the standard procedure is required on top of the topological routing network definition from the SVG file.

For instance, a specific turn can be forbidden to an a/c with a wingspan greater than 65m:

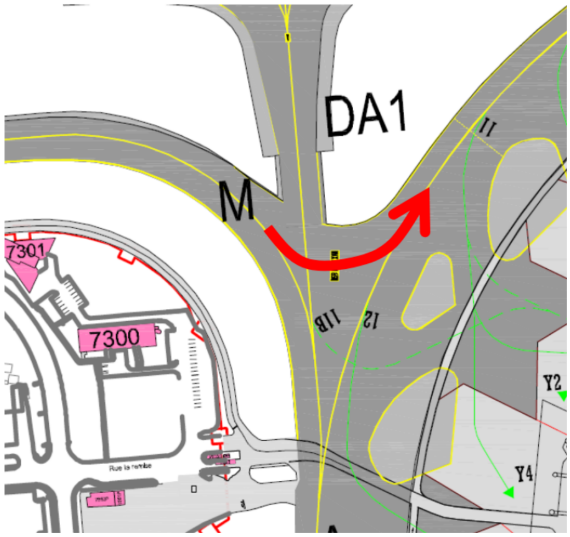

Figure 7: Operational traffic rule example

This information is described in an external JSON file.

There are two JSON parts:

- a first block to define paths that must respect the airport circulation rules, it constitutes a mapping of a path identification and the names of edges that constitute this path;
- a second block to define rules corresponding to previously defined paths.

It corresponds to the structure of this class diagram:

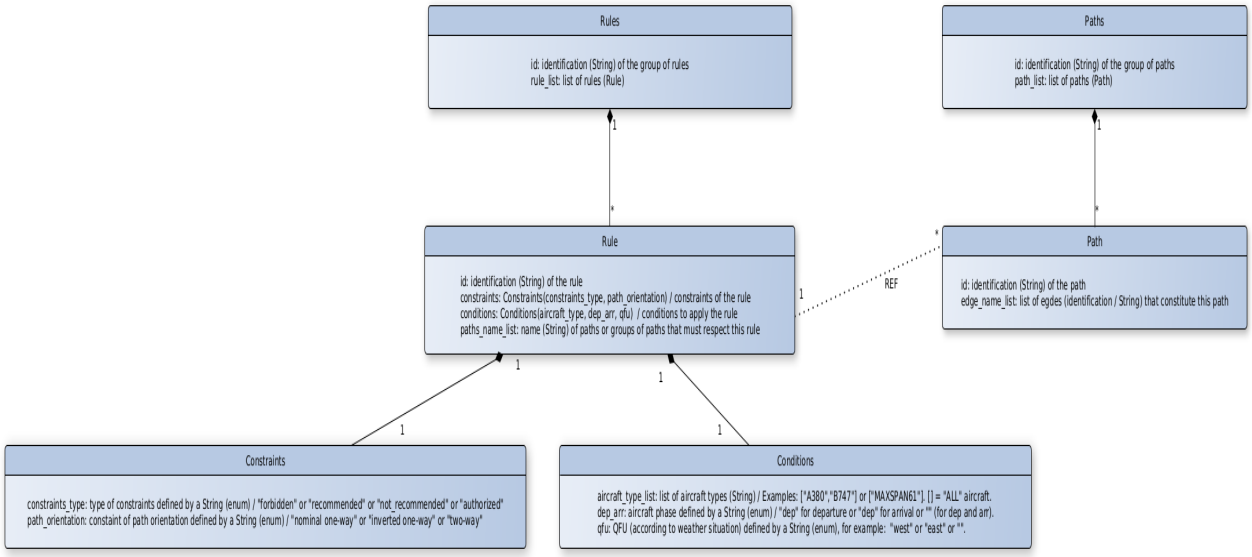

Figure 8: JSON traffic rules structure

```
{
    "id": "Paths_Def",
    "path_list": [
        {
            "edge_name_list": [
                "edge_idJ1-J41-1797",
                "edge_idJ1-J41-1807"
            ],
            "id": "J1"
        },
        {
            "edge_name_list": [
                "edge_idG-GE11-906",
                "edge_idE4-891",
                "edge_idE4-893",
                "edge_idE4-895",
                "edge_idE4-897",
                "edge_idE4-898"
            ],
            "id": "E4"
        },
        {
            "edge_name_list": [
                "edge_idGE12-917",
                "edge_idE5-919",
                "edge_idE5-935",
                "edge_idE5-944",
                "edge_idE5-959",
                "edge_idE5-947",
                "edge_idE5-945"
            ],
            "id": "E5"
        }
    ]
}
```


Figure 9: JSON block that defines rules

```
{
    "id": "Rules",
    "rule_list": [
        {
            "conditions": {
                "aircraft_type_list": [],
                "dep_arr": "dep",
                "qfu": "west"
            },
            "constraints": {
                "constraints_type": "recommended",
                "path_orientation": "inverted one-way"
            },
            "id": "departure_west",
            "paths_name_list": [
                "FR"
            ]
        },
        {
            "conditions": {
                "aircraft_type_list": [],
                "dep_arr": "",
                "qfu": "west"
            },
            "constraints": {
                "constraints_type": "recommended",
                "path_orientation": "inverted one-way"
            },
            "id": "manex_west",
            "paths_name_list": [
                "J1",
                "E4",
                "E5",
                "E12",
                "G12",
                "T",
                "RP",
                "RT",
                "R",
                "P",
                "W10",
                "Q",
                "B",
                "U",
                "Middle",
                "C"
            ]
        }
    ]
}
```


Figure 10: JSON block that defines paths

Details about constraints:

- constraints_type ("authorized" by default):
  - "forbidden" / "authorized”: hard constraint (mandatory / the rule cannot be violated)
  - "not-recommended" / "recommended": soft constraint (preferably / the rule can be violated)
- path-orientation ("two-way" by default):
  - "nominal one-way" (-->): only in one direction whose orientation is the same compared to that in the JSON description of paths
  - "inverted one-way" (<--): only in one direction whose orientation is inversed compared to that in the JSON description of paths
  - "two-way" (<-->): in two directions

Details about conditions:

- aircraft_type_list / several possibilities ("" = "ALL" by default):
  - Directly a list of aircraft type, for example: ["A230","B777"]
  - Or a list of categories of aircraft: like maximal span ("MAXSPAN61"...), ICAO aircraft classification ("A", "B"...), maximal take-off weight ("MTOW80"...) or other aircraft characteristics
- dep_arr ("" = dep and arr by default): aircraft phase
- qfu: configuration of ways defined by the airport (according to weather situation)

3.3 Simulation definition

The data generated for a simulation, i.e. aircraft flight plans and Taxibots missions and regulations in case of forged data for simulated AI, are described in a JSON format as explained in the following chapters.

3.3.1 Departure flights description

The first section of the file is the list of outbound flights. Basically it contains flight plan information.

```
"NewDepartureFlights": [
    {
      "CallSign": "DAL865",
      "AircraftType": "A319",
      "Dep": "LFPG",
      "Arr": "EHAM",
      "Rwy": "26R",
      "Parking": "Q10",
      "Ttot": "09:04:12",
```

```
      "Dlink": true,
      "Taxi": "taxibot",
      "StartTime": "08:35:00",
      "Tobt": "08:45:02"
    }, ...]
```


StartTime is the simulated time from which the radar plot will appear on the radar image. You can opt for two strategies:

- Make all the aircraft appear from the start of the simulation, by setting a starttime shorter than the earliest TOBT. Depending on the length of the exercise, this may not be realistic, as aircraft generally do not appear more than 10 minutes before TOBT.
- Make each aircraft appear a few minutes before its TOBT, which simulates the fact that the aircraft appears on the radar when the pilot switches on his transponder.

TTOT is not mandatory, it will be calculated using TOBT + taxi time.
The Rwy attribute can be the name of the runway, so the aircraft will be distributed to the different holding points according to their turbulence category.

3.3.2 Arrival flights description

The second section is the list of arrival traffic flight information. In the same manner, TIBT is not mandatory and will be recomputed from ETA.


```
"NewArrivalFlights": [
    {
      "CallSign": "AFR634",
      "AircraftType": "A320",
      "Dep": "EHAM",
      "Arr": "LFPG",
      "Rwy": "26L",
      "Parking": "H22",
      "Eta": "09:02:00",
      "Dlink": true,
      "Taxi": "set",
      "Tibt": "09:20:19"
    }, ...]
```


3.3.3 Taxibots missions

The third section deals with Taxibots.


```
"NewTaxibots": [
    {
      "CallSign": "TB01",
      "AircraftType": "TAXIBOT_NB",
      "Parking": "C03",
      "StartTime": "08:35:00",
      "Missions": ["FIN449", "KLM240", "DAL129"]
     }
```

```
    ]
```

The parking is the initial and final position of the Taxibot and the field 'Missions' list the aircraft (departure or arrival) it is supposed to tow.

3.3.4 Regulations to simulate AI

In the specific case of a simulated AI , the operator that prepares the data can manually set constraints on the aircraft trajectory in order to simulate an AI resolution to deconflict the trajectories.

Below is an example of regulation for an arrival aircraft:

```
      "Regulations" :{
          "Orders" : {"08:55:00": ["AircraftSpeedDown"]},
          "RwyCrossing": {"Hold" : "S1",
                          "Delay": 10},
          "RwyExit": "V2",


          "Coupling": {
               "Hold" : "SW1",
               "Duration": "00:01:30"
          }

      }
```

Orders is the list that can contain any pilot orders (defined in §4.3.3.5) with the simulated time at which it has to be executed.

RwyExit sets which exit the aircraft will use.

RwyCrossing sets the holding point the aircraft will cross the inner runway. Potentially a delay can be added so that the aircraft waits before crossing, avoiding a take off on the inner runway for instance. In the same manner for a departure aircraft, the runway entry can be forced (otherwise the aircraft will be set to different holds depending on their wake vortex category).

```
"LineUp": {"Hold" : "S1",
           "Delay": 10},
```

Coupling defines where and how long the taxibot coupling process will be, in case of a aircraft planned to be towed. The same exists for departure aircraft with *Decoupling* attribute, the coupling being done at parking.

Co-funded by the European Union

4 Distributed software architecture

4.1 IVY middleware

Ivy [2] is a simple protocol and a set of libraries and programs that allow applications to communicate with each other through text messages. Each connected application is called an agent. The background mechanism uses subscription based on regular expressions. Ivy libraries are available under an open-source licence (LGPL) in C, C++, Java, Python and Perl, on Windows, Unix and Macs. Several Ivy utilities and hardware drivers are also available. It has been used on several research projects and proved to be effective even when a large number of agents are involved. In addition it is simple to use and easy to set up.

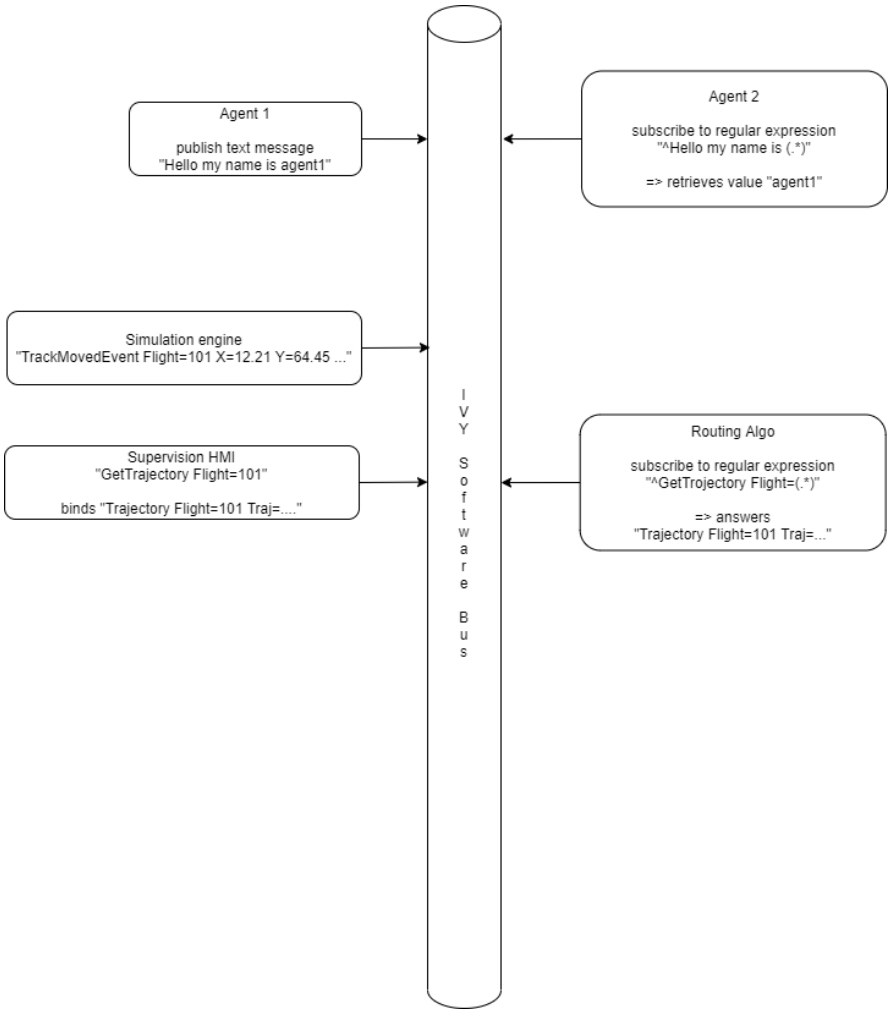

Figure 11: IVY principles

Ivy is currently used in research projects in the air traffic control and human-computer interaction research communities as well as in commercial products.

Actually, the IVY protocol is easy to use, and the data exchanged are human readable, which will facilitate the debugging. In addition, IVY has already proven to be capable to handle a heavy traffic, it is widely used at ENAC in ATC simulation with high traffic load.

4.2 Software agents

The simulation platform set up for ASTAIR validation is composed of the following software agents:

- Radar engine is in charge of emitting vehicles position and speed each second. It also recomputes trajectories in case of pilot orders, see §4.3.3.5. It is also responsible for simulation initialisation and flight information, see §4.3.3.1 and §4.3.3.3.
- Radar Image HMI is an A-SMGCS prototype developed over several projects where the ATCo can input the taxiing clearances. It cannot be rated at a single A-SMGCS level since it meets different levels depending on the functions:
    - Surveillance: Level 2. The radar image represents aircraft and tugs on the airport layout (apron and manoeuvring area) with their identity.
    - Control: Level 2. Control function can detect any conflict concerning mobiles on the movement area. The alarms are provided to the controller.
    - Route Planning: Level 4. The route planning function determines the best route for an aircraft on the ATCo's request. The calculated route considers the ground rules and other traffic. The validated route can then be electronically sent to the pilot if the aircraft is equipped with datalink.
- Supervision working position is composed of 2 HMIs. The first HMI shows the incoming flights and distribution of flights management between human and AI. The second HMI is an adapted version of radar image that allows the supervisor to look into the future.
- Fleet Manager HMI shows the tugs allocation planning that has been decided by the AI system.
- The core AI of ASTAIR solution is a Multi Agent System that can computes vehicles trajectories over 20 minutes in the future and deconflicting them with speed regulations. It also manages the mission of Taxibot tugs.
- Scenario orchestrator is a simple agent that automatically executes actions in a timely manner according to simulation time in order to make the simulation more realistic. In addition, this agent is used in the simulated AI mode to 'automatically' regulate aircraft to avoid conflicts.
- Interpretation agent tries to detect the speed regulations decided by AI in the planned trajectories because the Multi Agent System, as most AI systems, is a black box which provides the deconflicted trajectories but cannot explain how its decisions were made.

4.3 Data exchanges during simulation

The following chapters describe the data exchanged during a simulation exercise.

4.3.1 Setup with integrated Multi Agent System

The diagram below presents the main data flows between the different parts of the ASTAIR simulation. In the full integration mode with MAS, versus simulated AI, the data preparation phase consists in defining:

- the airport map with a sufficient quality level for routing purposes,
- the flight schedule that defines the traffic load and the use case scenario
- the scenario file if some actions need to be automated to fit the use case.

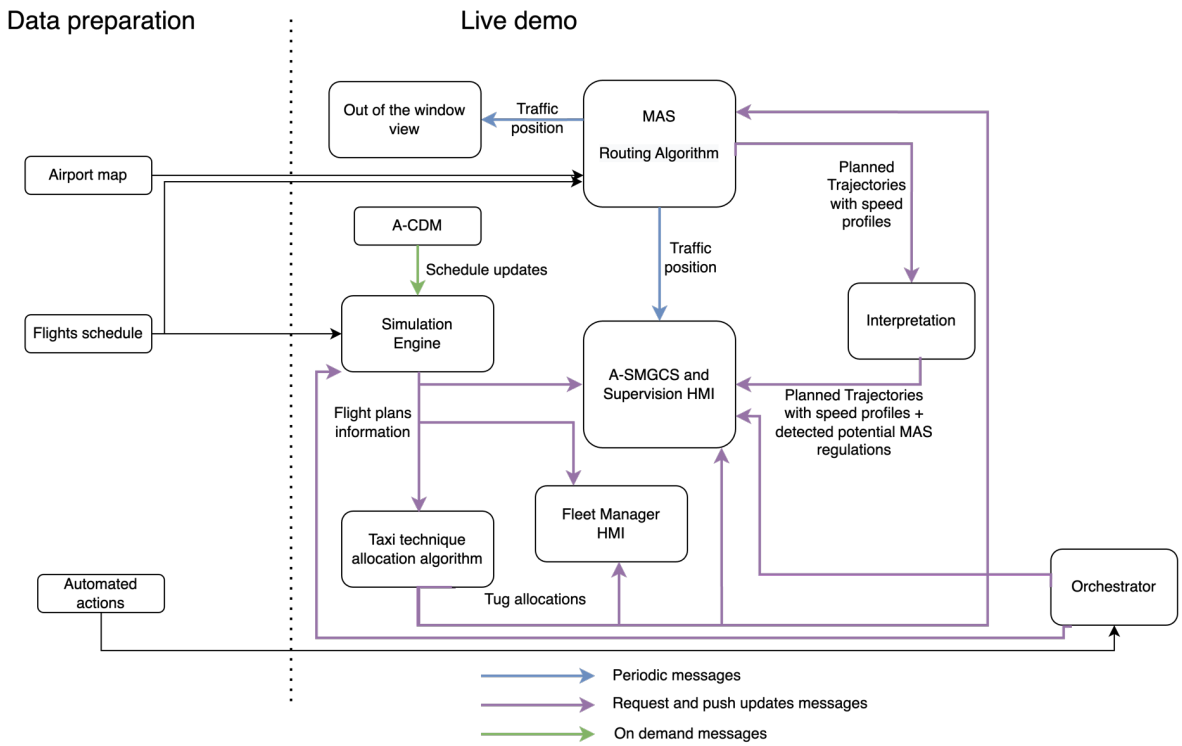

Figure 12: Setup with integrated Multi Agent System

4.3.2 Setup with simulated AI

The main difference in the setup with simulated AI is on the data preparation. Indeed, everything that will happen in the scenario is pre-computed and manually adjusted from vehicles trajectories to tugs allocations. This mode cannot be used in use case demonstration that requires live modification of the plan, i.e. it cannot be used for re-scheduling case, but it is very powerful for HMI designs workshops.

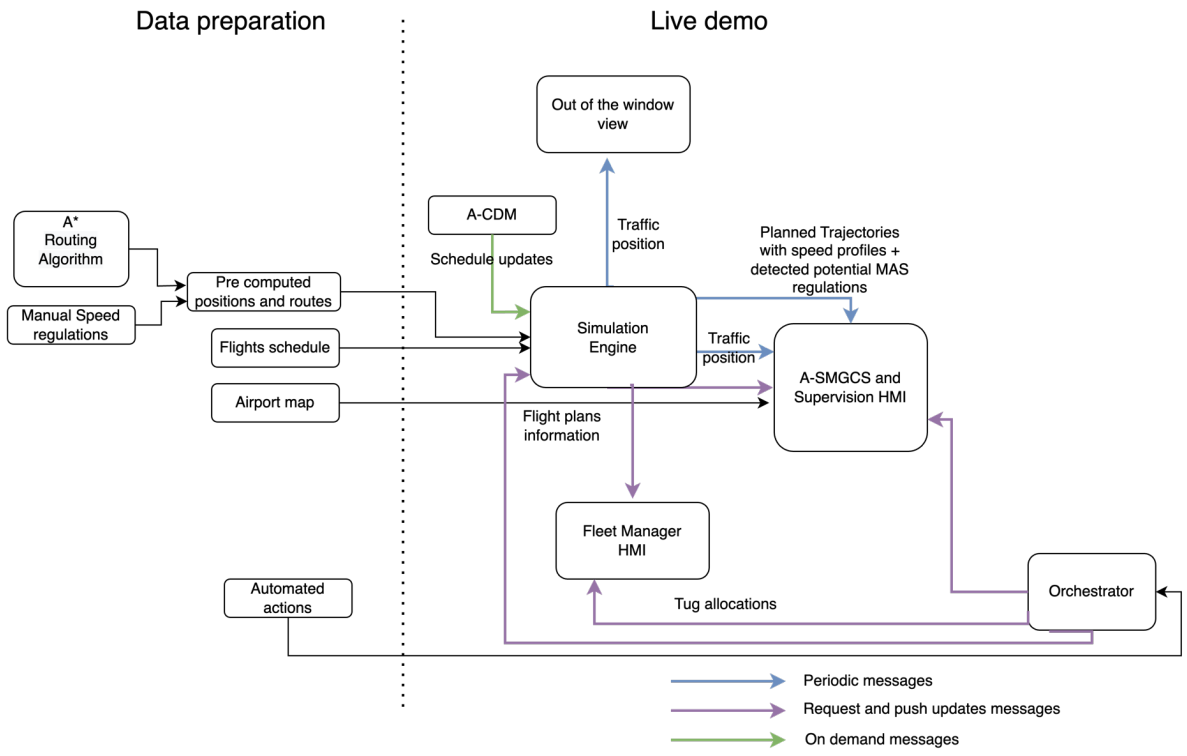

Figure 13: Setup with simulated AI

4.3.3 IVY messages

4.3.3.1 Simulation initialization

At execution start-up, an IVY agent will usually follow the sequence below:

1. Send a message to get the list of the simulated vehicles, either GetDatabaseInfos or GetCurrentFlights
2. It will receive a list of identifiers and for each one will send a request for its position (GetGroundPosition), its flight plan (GetPln) and its taxi technique (GetTaxiMethod)
3. Then during the simulation, these data will be updated via event messages.

4.3.3.1.1 GetDatabaseInfos

Message sent to Simulation Engine to get all the flights id present in the database.

Message: “GetDatabaseInfos MsgName Cond=CallSign=*”

The response is “DatabaseInfos MsgName Nb= List=” with ‘nb’ the number of flights present in the database and ‘List’ a list with all their id.

The Simulation Engine will give all the flight identifiers used in the simulated traffic, past, present and future.

4.3.3.1.2 GetCurrentFlights

Message sent to Simulation Engine to get all flights id who are used at the time asked in the message.

Message: “GetCurrentFlights MsgName Time=”

The response is: “CurrentFlights MsgName Time= List=” with the same time and the list with all flights ids present at that time.

Only the flights that are currently active are listed in the response.

4.3.3.1.3 GetTaxiMethod

The GetTaxiMethod message is sent to Simulation Engine for a given aircraft to get its taxi method at the current simulation time.

Message: “GetTaxiMethod MsgName Flight=”

The response is: “TaxiMethod MsgName Flight= Taxi= Tug=”

where Taxi can be classic, set (single engine taxiing), e-taxi, wheel tug or taxibot and Tug is the id of the coupled tug if the current taxi method is taxibot.

4.3.3.2 Clock messages

These messages are used to handle the simulation time flow.

4.3.3.2.1 ClockStart

Message sent to Simulation Engine by another application to restart the simulation clock.

4.3.3.2.2 ClockStop

Message sent to Simulation Engine by another application to stop the simulation clock.

4.3.3.3 Flights information

Any agent can ask the Simulation Engine to get the flight plan or update its content if necessary.

4.3.3.3.1 GetPln

Message: “GetPln MsgName Flight= From=now”

The response provides all information for the given flight with the message:

“Pln MsgName Flight= Time= CallSign= AircraftType= Ssr= Speed= Rfl= Dep= Arr= Rvsm= Tcas= Adsb= Dlink= Proc= Rwy= Pkg= PossibleTmos= PlannedTmo= Tibt= Tobt= Ctot= Ttot= Eta= List=”

4.3.3.3.2 SetPln

Any agent can update one or many flight plans’ data by sending this message:

Message form: “SetPln Flight= Time= PlannedTmo= Eta= Tibt= Tobt= Ttot= Ctot= Pkg= List=”

In this message, it just needs to fill the field to be updated.

4.3.3.3.3 SetSectorIn

The SetSectorIn message is sent to Simulation Engine to set the sector or the frequencies of a given aircraft at a given time.

Message: “SetSectorIn Flight= Sector= Time=”

Simulation Engine will automatically send a SectorEvent if necessary (if the time is equal to the current simulation time).

4.3.3.4 Periodic messages

4.3.3.4.1 ClockEvent

The ClockEvent message is sent by Simulation Engine every second to notify all agents the current simulated time.

Message: “ClockEvent Time= Rate= Bs=0” where the time is the simulated time, rate is the rhythm of the simulation (=1 in normal simulation) and Bs=0 is corresponding to a normal simulation

4.3.3.4.2 TrackMovedEvent

The TrackMovedEvent is sent by Simulation Engine to notify the position and others information for a given flight at a given time. In our simulator, this message is automatically sent every second for all used flights.

Message: “TrackMovedEvent Flight= CallSign= Ssr= Sector= Layers= X= Y= Vx= Vy= Afl= Alti= Rate= Yaw= Pitch= Roll= GroundSpeed= Tendency= EdgeId= NextNode= DistNextNode= TlastNode= Time=”

with specific information needed for the routing dependant to the layout and the routing network (edge id, next node, distance to the next node, and the time where the aircraft passed the last node).

4.3.3.4.3 TrackDiedEvent

The TrackDiedEvent is sent by Simulation Engine automatically after a few seconds where the simulator has no more information about its position. For example, when a taxibot is attached to an aircraft, a track died event is send for this given taxibot.

Message: “TrackDiedEvent Flight=”

4.3.3.4.4 SectorEvent

The SectorEvent is automatically sent by Simulation Engine when the aircraft will change its sector or its frequency.

Message: “SectorEvent Flight= SectorOut= SectorIn=”

4.3.3.4.5 BeaconEvent

The BeaconEvent is automatically sent by Simulation Engine when an aircraft pass over a beacon (for an aircraft, it corresponds to a holding stop or a gate).

Message: “BeaconEvent Flight= Beacon= Fl= Mode= Time=” where Fl is the flight level (0 on the airport), and mode is vertical or abeam (only useful for airborne a/c)

4.3.3.4.6 TaxiwayEvent

The TaxiwayEvent is automatically sent by Simulation Engine when an aircraft changes taxiways. This message is useful for A-SMGCS HMI to display this information.

Message: “TaxiwayEvent Flight= Taxiway=”

4.3.3.4.7 PlnEvent

The PlnEvent is automatically sent by Simulation Engine when the flight plan is changed, exactly after each SetPln received.

Message: “PlnEvent Flight= Time= CallSign= AircraftType= Ssr= Speed= Rfl= Dep= Arr= Rvsm= Tcas= Adsb= Dlink= Proc= Rwy= Pkg= PossibleTmos= PlannedTmo= Tibt= Tobt= Ctot= Ttot= Eta= List=”

4.3.3.4.8 TaxiMethodEvent

The TaxiMethodEvent is sent by Simulation Engine when the current taxi method changes during the simulation, e.g. when the pseudo pilot starts eTaxi.

Message: “TaxiMethodEvent Flight= TmoOld= TmoNew=”

4.3.3.5 A/C pilot orders

Some messages are used to modify aircraft behaviour in the simulation, all these messages are received and handled by Simulation Engine. When Simulation Engine processes a pilot order, it recomputes the future positions of the given aircraft that will be sent to the TrackMovedEvent messages.

4.3.3.5.1 SetTaxiMethod

The SetTaxiMethod message is used to set the current taxiing method of a given flight.

Message: “SetTaxiMethod Flight= Tmo=”.

After receiving this message, Simulation Engine sends a TaxiMethodEvent message as described above.

4.3.3.5.2 AircraftTaxi

The AircraftTaxi message is sent to Simulation Engine to have an aircraft taxi on a given route.

Message: “AircraftTaxi Flight= Airport= Stop= To= Edges=”

where ‘Airport’ is the airport of the given flight, ‘Stop’ is the final point on taxiing route, ‘To’ is the type of final point and ‘Edges’ the list of edges id defining the route to follow, read in the airport SVG map.

4.3.3.5.3 AircraftPushback

The AircraftPushback message is used to pilot an aircraft to taxi on its given pushback path.

Message: “AircraftPushback Flight= Airport= Stop= Edges=”

where ‘Airport’ is the airport of the given flight, ‘Stop’ is the final point on taxiing route and ‘Edges’ the pushback route.

4.3.3.5.4 AircraftSpeedUp or AircraftSpeedDown

The AircraftSpeedUp and AircraftSpeedDown messages are used to speed up or speed down a given aircraft by steps of five knots.

Message: “AircraftSpeedUp Flight=” or “AircraftSpeedDown Flight=”

4.3.3.5.5 AircraftSetSpeed

The AircraftSetSpeed message is used to set up the speed for a given aircraft .

Message: “AircraftSetSpeed Flight= Target=”

4.3.3.5.6 AircraftStopTaxi

The AircraftStopTaxi message is used to stop a given aircraft.

Message: “AircraftStopTaxi Flight=”

4.3.3.5.7 AircraftResumeTaxi

The AircraftResumeTaxi message is used to resume the route of a given aircraft after an AircraftStop message.

Message: “AircraftResumeTaxi Flight=”

4.3.3.5.8 AircraftLineUp

The AircraftLineUp message is used to line up a given aircraft on a specific runway.

Message: “AircraftLineUp Flight= Rwy=”

4.3.3.5.9 AircraftTakeOff

The AircraftTakeOff message is used to take off a given aircraft on a specific runway.

Message: “AircraftTakeOff Flight= Rwy=”

4.3.3.5.10 AircraftRollingTakeOff

The AircraftRollingTakeOff message is used to line up and take off a given aircraft on a specific runway.

Message: “AircraftRollingTakeOff Flight= Rwy=”

4.3.3.6 Tug drivers orders

Similarly to the pilot orders, the following messages concern the handling of tugs.

4.3.3.6.1 AttachAircraft

The AttachAircraft message is used to attach a taxibot to an aircraft.

Message: “AttachAircraft Flight= Tug=” with the flight id and the tug id.

4.3.3.6.2 AircraftAttaching

The AircraftAttaching message is sent to the out of the window view HMI, A-SMGCS and Fleet Manager HMI to start the animation of coupling.

Message: “AircraftAttaching Flight= Tug=”

4.3.3.6.3 AircraftAttached

The AircraftAttached messge is used to tell the aircraft is attached to the taxibot at the end of the coupling phase animation from the out of the window view HMI.

Message: “AircraftAttached Flight= Tug=”

4.3.3.6.4 DetachAircraft

The DetachAircraft message is used to detach a taxibot from an aircraft.

Message: “DetachAircraft Flight= Tug=” with the flight id and the tug id.

4.3.3.6.5 AircraftDetaching

The AircraftDetaching message is sent to the out of the window view HMI, A-SMGCS and Fleet Manager HMI to start the animation of decoupling.

Message: “AircraftDetaching Flight= Tug= TugPosition=” with TugPosition the future position of the taxibot after decoupling phase.

4.3.3.6.6 AircraftDetached

The AircraftDetached message is used to tell the aircraft is detached from the taxibot at the end of the decoupling phase animation from the out of the window view HMI.

Message: “AircraftDetached Flight= Tug= TugPosition=” with TugPosition the position of the taxibot after decoupling phase.

4.3.3.7 Ground operations supervision

The messages described in this section are dedicated to ASTAIR needs and AI supervision.

4.3.3.7.1 GetMovingTracks

The GetMovingTracks message gives the list of moving vehicles in the timeframe between Start and End parameters.

Message: “GetMovingTracks MsgName= Start= End=”

Answer: “Moving Tracks <MsgName> List=”

With ‘List’ being a comma separated list of flight ids.

4.3.3.7.2 GetFuture4DTraj

The GetFuture4DTraj message gives the detailed trajectory as planned by the AI multi agent system for a given flight.

Message: ”GetFuture4DTraj MsgName= Flight=”

Answer: ”Future4DTraj <MsgName> Flight=<id> Data=”

With Data being a list of data points composed of Time, X, Y, Heading Vx, Vy information.

4.3.3.7.3 TimeForClearance

TimeForClearance messages are used to keep the supervisor up to date with operations schedule for a flight. For now, it used to give the push back time and for towed aircraft or single engine taxiing aircraft, to give an estimated engine start up time that could be provided to pilots.

Message: “TimeForClearance Type=[Engine|PushBack] Flight=”

4.3.3.7.4 GetMASAircraftPriorities

GetMASAircraftPriorities message aims at providing the supervision HMI with the priorities between aircraft decided by AI. It will be used to try an interpretation of the computed solution.

Message: “GetMASAircraftPriorities MsgName=”

Answer: “MASAircraftPriorities <MsgName> List=”

4.3.3.7.5 GetRegulations

GetRegulations requests are used by supervision HMI to ask for explanations on MAS computed results. As explained in §4.2, the AI does not give explanations on how the routing solution has been computed. Explanation agent will try to find basic explanations and provide them with this message. The result is formatted in JSON format that matches the format explained in §3.3.4.

Message: “GetRegulations MsgName=”

Answer: “Regulations <MsgName> Data=<json answer>”

4.3.3.7.6 GetPath

The GetPath message is used to ask a path suggestion for a given flight from its position to a destination point via some points if the ATCO want to edit himself the path.

Message: “GetPath MsgName Flight= From= Via= To=”.

The ‘To’ field can be equal to ‘default’, meaning the routing has to find the best destination point depending to the traffic.

The response sent by the routing agent is defined as follow:

Response: “Path MsgName Flight= Path= SpeedProfile=”

with SpeedProfile a list of speed evolution in time.

5 Data logging

In addition to observations and manual counting of events by the team, the information to be logged during the final evaluation exercises are the following:

- Number of changes in path suggestion by ATCO
- Time spent on ATC and Pilot frequencies
- Duration of information exchange over radio b/w ATC and Pilot/drivers
- Number of Telephone communication b/w ATCO and FM
- Number of Radio communication b/w ATCO and Pilots
- Number of radio communication for each aircraft
- Percentage of time spent in frequency ATCO over the run
- Time spent by ATCOs on telephone
- Number of speed target has been modified per aircraft (i.e. number of re-computations triggered by MAS algorithm)
- Taxiways occupancy (average number of aircraft taxiing at the same time)
- Specific metrics related to the interaction with the HMI (number of selections, number of uses of each feature and interactions…)

6 List of acronyms

The following table reports the acronyms used in this deliverable.

Term	Definition
ACHIL	Aeronautical Computer Human Interaction Lab
AEON	Advanced Engine Off Navigation
AI	Artificial Intelligence
AMAN	Arrival MANager
A-SMGCS	Advanced Surface Movement Guidance and Control System
ASTAIR	Auto Steer Taxi at Airport
ATC	Air Traffic Control
ATCO	Air traffic Controller
ATM	Air Traffic Management
EASA	European Aviation Safety Agency
ETA	Estimated Time of Arrival
HMI	Human Machine Interface
JSON	JavaScript Object Notation
MAS	Multi Agent System
SVG	Scalable Vector Graphics
TCAS	Traffic Collision Avoidance System
TIBT	Target In Block Time
TOBT	Target Off Block Time
TRL	Technology Readiness Level
TTOT	Target Take Off Time
MAS	Multi Agent System

7 References

[1] RealTwr - https://realtwr.fr/

[2] IVY Software bus - https://www.eei.cena.fr/products/ivy/ and https://gitlab.com/ivybus/ivy-python

[3] X-Plane filght simulator - https://www.x-plane.com/

[4] X-Plane Scenery Gateway - https://gateway.x-plane.com/

[5] X-Plane APT file specification - https://developer.x-plane.com/article/airport-data-apt-dat-file-format-specification/

[6] Open Topo Data - https://www.opentopodata.org/

[7] A* Algorithm https://en.wikipedia.org/wiki/A*_search_algorithm