Mercury: an open source platform for the evaluation of air transport mobility

Luis Delgado, Gérald Gurtner, Michal Weiszer, Tatjana Bolić, Andrew Cook
Centre for Air Traffic Management Research, School of Architecture and Cities
University of Westminster
London, United Kingdom
[g.gurtner|l.delgado]@westminster.ac.uk

***Abstract*—The Mercury simulator is a platform developed over several years during exploratory research projects. It features a detailed description of the air transportation system at the European level, including passengers and aircraft, plus various important actors such as the Network Manager, airports, etc.**

This article presents the possibilities offered by the simulator's last and now open-source version. We describe the core Mercury functionalities and highlight its modularity and the possibility of using it with other tools. We present its new interface, which supports user-friendly interaction, exploring its data input/output and setting its various parameters. We emphasise its possible uses as a solution performance assessment tool, usable early in the innovation pipeline to better estimate the impact of new changes to the air transportation system, particularly with respect to other system components. We hope opening the simulator may encourage other models to become available, allowing faster prototyping of SESAR Solutions early in the innovation pipeline and an *in fine* standardisation and higher performance of simulation-based performance assessment tools.

***Keywords*—Air transportation, agent-based model, performance assessment, simulator, open-source, passengers, airline cost model, simulation as a service**

Evaluating the performance of a complex system like air transport is a challenging task. Technological solutions tend to be assessed on a subset of the system, making it difficult to assess their network-level impact, especially when considering the interactions with other mechanisms.

The system performance is driven by uncertainty, disruptions, and, importantly, by the interaction of many different actors, which tend to make decisions in sub-optimal conditions: reacting to disruptions, estimating uncertainties in the system and without full knowledge of the intention of other actors. Agent-based modelling (ABM) can tackle these types of systems. The decisions of each agent (element in the system) can be modelled individually with relatively simple rules. Still, an emergent behaviour arises when the agents interact in an evolving environment capturing the richness of the network interactions in the system.

Previous research has proven that passenger delay needs to be modelled to assess the system's performance besides the 'nominal' Key Performance Indicators (KPIs) used by the ATM industry, which have a strong flight-centric focus. This is relevant for different reasons: first, passenger delay translates into cost (*e.g.* due to passenger compensation Regulation 261 [1]) and therefore drives some of the behaviour of airlines; and second, as passengers experience delays different than flights due to their connections [2] and the fact that they need to reach their final destination in a door-to-door journey [3], which might include multi-modal trips [4].

Therefore, there is a need for a platform to evaluate ATM solutions and policies flexibly (modifying agents' behaviour or assessing external systems), including the previous considerations. The Mercury simulator has been developed in this context for over ten years. Throughout different projects, a first simulator version able to capture passenger-centric indicators [2] was expanded to consider cost resilience [5] and door-to-door journeys [6]. The simulator was redesigned as a full agent-based model enabling the development of new metrics to capture network effects [7]. In recent projects, work has been devoted to easing the integration of external modules and solutions (*e.g.* supporting human-in-the-loop simulations with interaction with an external interface [8]). Mercury incorporates processes and behaviours of actors to the environment considering their expected cost functions. Hence, it provides not only a technological and procedural evaluation platform but also the possibility for policy evaluation (*e.g.* changes to passenger compensation through EU Regulation 261 [1]).

In contrast with other available tools, Mercury captures the emergence of ATM performance from the behaviour of the different agents, which react based on expected operational costs. This enables the capability of modelling the decision-making processes of key actors (particularly airlines) and, furthermore, not only the evaluation of (technical) Solutions but also policy changes (as they would translate into different expected costs and hence decisions when managing flights). Mercury, provides, in addition to *standard* ATM KPIs, cost and passenger-related indicators by tracking individual passenger itineraries, including their connections. The event-driven approach of Mercury enables a fast-time simulation of a day of operations in the whole of the ECAC region in a few minutes. This is required due to Mercury's stochasticity, as more than one run will usually be needed to obtain statistically significant results.

Other platforms available tend to be flight-centric and lack cost and passenger modelling. This makes it difficult for them to model the actors' decisions properly and cannot extract the low-level KPI distributions available from Mercury. However,

they can, in some cases, provide higher performance on the evaluation of particular flight-centric aspects of ATM, *e.g.* EUROCONTROL's R-NEST provides highly detailed airspace and capacity management modelling[1], or the open platform BlueSky [9] provides detailed trajectory integration on time-based simulations of flights, which would be required for the modelling of conflict detection and resolution tools, which are too low-level to be captured by Mercury.

Mercury has successfully been applied in a range of challenges, such as the evaluation of the alignment of KPIs at the European level [10], assessment of SESAR solutions with consideration of network-level interactions [7], analysis of emergent behaviour on delay management [11], and the analysis of UDPP concepts [12]. It has proven to be a powerful simulator to capture the interaction between air traffic actors, focusing on estimating performance indicators of airlines, flights and passengers.

Given the need for European-wide assessment tools that support multi-KPI estimations on one hand and the capabilities of Mercury on the other, it was decided to open the code source of Mercury, to make it available to the wider research community.

Hence, we present in this article a short description of the various workings of the simulator, its main strengths, how new users could use it – for instance, developing new modules/add-ons for it or interfacing with existing models – and the type of interface they may expect to set up their simulations and explore such results. It is therefore intended as a technical communication paper to present a synthesis of the results of the simulator development over many years, with a view to its further development, rather than presenting the experimental results of running the simulator in various projects[2].

I. Open source Mercury

Mercury has reached a maturity level at which it can be made open source to facilitate the usage of the simulator by the community. This will enable students, researchers and practitioners to support the core model's development and integrate their modules and systems to transform Mercury into a truly air transport mobility performance analyser.

A. Programming Language and libraries

Mercury is developed in Python 3.10, a very popular, free and open-source language. Python uses an interpreter at runtime, which makes it platform-independent albeit generally slower than compiled languages such as C++.

The development has been done from scratch but relies on third-party libraries like Simpy[3], for the discrete-event simulation framework, pandas[4], for data structure and manipulation, and NumPy[5], for fast numerical computations on arrays and matrices. These libraries are all open-source.

Besides the Python packages used for the development of Mercury, the model uses the following external libraries:

- Hotspot[6]: Developed as part of BEACON project [8] to manage hot spots and air traffic flow management (ATFM) regulations for UDPP concepts. Open-source under GPL v3.
- uow-tool-belt[7]: Library providing basic tools and functionalities *e.g.* read/write functionalities. Open-source under GPL v3.
- Aircraft performance: Library with the implementation of some performance models. Proprietary.

The aircraft performance data from EUROCONTROL's BADA model [13] is used for the aircraft performance library. Any user will thus need to require a licence from EUROCONTROL, normally granted free of charge for researchers.

B. License and distribution

Mercury is distributed under a GPL v3 licence, which implies that any distribution of part of its code has to be done under the same terms. The code is available on GitHub[8].

C. Data manipulation and system modification

To facilitate the input and output manipulation, Mercury uses the open source Apache Parquet format[9][10]. Data is compressed efficiently in tables that are easy to analyse and manipulate while maintaining properties such as their data types. See Section VI for more information about the data used and generated by Mercury.

Configuration files are encoded using TOML format[11] which is easy to read by humans and easy to integrate and parse in Python.

II. Model specification and design

A. Model requirements

Mercury aims at capturing the interaction between ATM elements to provide performance indicators for key stakeholders. In particular, the system should be able to:

- Model the pre-departure and tactical operations of flights and passenger mobility in Europe.
- Test ATM solutions, which modify procedures and other rules. The model should facilitate the changing of these rules and the integration of new elements.
- Compute KPIs for stakeholders. In particular, delay distributions for passengers, flights and costs.
- Capture knock-on effects between subsystems.
- Consider fairly complex decision-making processes from the airlines when reacting to cost and not only to delay.

The model needs to consider the most important channels of propagation of delay. First, aircraft are used throughout the day by different flights and thus can propagate delays

[1] https://www.eurocontrol.int/solution/rnest (accessed November 2023)

[2] Which, although variously cited for reference herein, would be too extensive to capture in these nine pages.

[3] https://simpy.readthedocs.io/ (accessed September 2023)

[4] https://pandas.pydata.org/ (accessed September 2023)

[5] https://numpy.org/ (accessed September 2023)

[6] https://github.com/andygaspar/Hotspot

[7] https://github.com/UoW-ATM/uow_tool_belt

[8] https://github.com/UoW-ATM/Mercury

[9] https://parquet.apache.org/ (Accessed September 2023)

[10] Mercury allows the use of other formats as source (csv, MySQL database).

[11] https://toml.io/en/ (accessed September 2023)

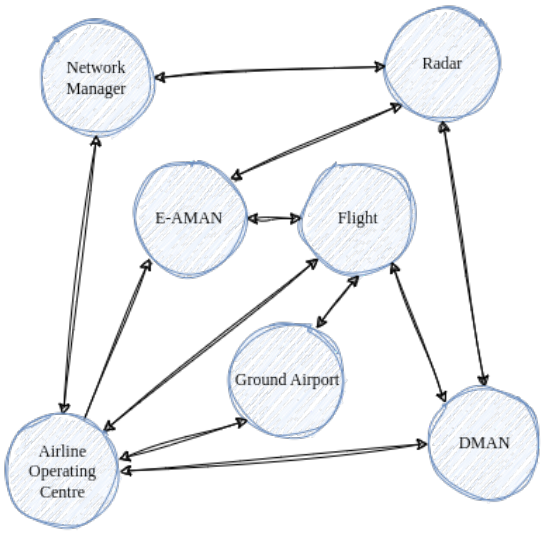

Figure 1. Acquaintances between agent types, derived from interactions between their underlying roles.

(reactionary delay). Second, passengers must be modelled to capture the costs of their delays and missed connections. Third, the model needs to consider exogenous sources of delays, *e.g.* turnaround delays. Fourth, the system's reactions to delays and/or congestion must be included, *e.g.* airlines' actions. Lastly, the model needs to allow dynamic decisions from agents. This dynamic aspect is paramount since operations face daily uncertainty, which triggers decisions that have knock-on effects on the system.

B. Scope

The Mercury simulator focuses on evaluating a day of operations, including pre-departure processes and tactical operations of flights and passengers' itineraries. Whilst the simulator is generic, some of its elements (particularly costs) are designed to work with the particularities of the European system.

C. Development methodology

The Gaia methodology has been used to build the simulator, defining roles and their interactions and grouping them into agents which provide services [14].

A role can be seen as an abstract description of an entity's expected function, defined by its responsibilities, permissions, activities and protocols. 40 roles are identified in Mercury, such as the 'flight planner', in charge of generating and selecting flight plans to be used, and the 'taxi-out estimator', to estimate the taxi-out time of a flight, or the slot assigned to assign an arrival slot by an AMAN. See [15] for a detailed description of all forty roles.

Roles are grouped into higher-level entities, which represent the agent types. The roles' activities become the agents' activities, and the roles' protocols are turned into the agents' services. The agents' internal roles can directly access all information available to an agent, while agents communicate using messages between them to request data and computations. In Mercury, the process of defining these agents was guided by existing entities in the ATM domain, such as the Airline Operating Centre (AOC). Seven agent types are defined in Mercury as depicted in Figure 1, which also presents the visibility between agents showing the centrality of the AOC.

III. Agents description

A. Airline Operating Centre

Mercury has one 'Airline Operating Centre' (AOC) per airline. They perform:

- the fleet management (*i.e.*, dispatching processes), including the selection of flight plan, flight cancellation and flight plan adjustments (*e.g.* waiting for (connecting) passengers, dynamic cost indexing).
- the tactical reassignment of passengers to flights if they need to be reallocated due to missed connections.

The estimation of expected costs drives the decisions performed by the AOC.

Cost modelling: Mercury considers direct operating costs of the flights based on the selected flight plan (*i.e.*, en-route charges and expected fuel usage); and costs due to unforeseen circumstances during the operations, in particular, 'cost of delay' (the extra cost that airlines experience due to the delay of their flights).

One of the main drivers of tactical decisions is the reduction of passenger-related costs. Mercury models both 'hard' costs, having a direct monetary translation, *e.g.* duty of care and compensations as per Regulation 261 [1], and 'soft' costs, of a future loss due to passengers' dissatisfaction and the subsequent impact on the market share.

Other operational costs, such as curfews, are also modelled as early delays can translate to reactionary delays, eventually breaching a curfew. Therefore the behaviour of the airline might vary to consider this.

Modelling passengers: Passenger groups are modelled as simple placeholders which contain information on the passenger itineraries and their characteristics, which other agents, notably the AOC, handle.

Response to events: The AOC listens for and executes actions when the following events are triggered:

- 'Flight plan (FP) submission': created at simulation initialisation and triggered 3 hours before each flight's scheduled off-block time (SOBT). When the event is triggered, the AOC chooses an FP. The selection of FP is an iterative process with the submission to the Network Manager (NM) and the potential reception of ATFM delay, which could trigger reconsidering which FP to use.
- 'Delay estimation': triggered one hour before estimated off-block time (EOBT). The AOC gets notified of non-ATFM delay (if any) and reassesses its estimated departure time. If the flight has an ATFM slot which will be missed, a new slot is requested from the NM. If the delay exceeds 30 minutes, a new flight plan is recomputed.

- 'Passenger check': 5 minutes before EOBT, the AOC computes the expected arrival time of connecting passengers that are supposed to board this flight and are not there yet, and decides whether to wait for them.
- 'Push-back': At the push-back event, the AOC checks which passengers are on board and computes the passengers' metrics such as departing delay, type of delay, etc. For passengers who missed a connection but are already at the airport, a reallocation process is triggered based on itineraries, aircraft space, fares, and ultimately, assigning Reg. 261 compensation.
- 'Flight arrival': This is triggered when a flight arrives at the inbound gate. The AOC starts two parallel processes: aircraft turnaround and passenger connection processing.

B. Flight agent

'Flight' agents integrate flight movements (ground and trajectory). They also capture the actions performed by the crew *e.g.* requesting a departing slot. There is one Flight agent per flight in the simulation.

Response to events: The Flight listens to the following events:

- 'Push-back ready': The Flight requests from the Departure Manager a departure slot and, considering an estimated taxi-out time provided by the Ground Airport agent, computes the push-back time. Congestion at the departure might produce some delay.
- 'Push-back': The Flight requests the actual taxi-out time from the Ground Airport and calculates the take-off time.
- 'Take-off': The Flight starts the trajectory integration modelling uncertainties associated with the weather (wind) and flight distance (*e.g.* due to uncertainty for ATC interventions). As part of this trajectory integration, the flight triggers the 'Flight Crossing Point' to notify the Radar agent of the flight progression. The Flight agent records the information related to the flight performance, *e.g.* fuel usage.
- 'Landing': The Flight requests the taxi-in time from the Ground Airport and triggers the 'Flight arrival' event.

A flight can provide information to other systems, such as estimated landing time, and add constraints to the flight plan to meet tie-window requests *e.g.* to land a slot provided by the E-AMAN.

C. Ground Airport agent

The 'Ground Airport' agents process arriving passengers (computing the actual transfer time between flights in the terminals) and the arrival of flights (providing estimated and actual turnaround times). Both processes rely on probabilistic modelling of their distributions considering characteristics of operations such as airport, airline, aircraft and passenger types. One Ground Airport agent is instantiated per airport.

D. DMAN and E-AMAN agents

Each airport has associated 'DMAN' and 'E-AMAN' agents to manage the departure and arrival queue of slots needed to respect the runway capacities, which can be adjusted and vary through the day. DMAN is the simplest agent; nominal capacity values incorporating the average effects of mixed operations and aircraft sizes are considered. For the E-AMAN, the capacities are defined based on the information on airport capacities, and they might be adjusted if ATFM regulations are issued at the airport. All airports have an E-AMAN to manage the arrivals; in some, a planning horizon for trajectory adjustment and reduction of holding time is also implemented.

E. Radar agent

The 'Radar' agent broadcasts the flight position to all interested agents, as enabled by a flexible subscription notification architecture. There is only one Radar agent.

At the initialisation of the simulation, interested agents register to the Radar and ask to be notified when a flight of certain characteristics (*e.g.* arriving at a given airport) verifies a given condition (*e.g.* reaching a point before landing).

During the FP submission, the Radar agent receives the FP from the NM, and creates an *augmented* flight plan adding the required 'Flight Crossing Point's considering the registered request for notification,

Response to events: The following events trigger the Radar:

- 'Flight Crossing Point': By crossing significant waypoints, the Flight triggers 'Flight Crossing Point', and the 'Radar' notifies subscriber agents of this event.

F. Network Manager (NM) agent

The Network Manager has a simplified view of the European airspace, and only one instance of NM exists in the simulation. It does not have an explicit knowledge of the airspace. Instead, the NM uses:

- random en-route ATFM delays, based on empirical data,
- regulations at airports modelled as 'queue's of slots.

The NM processes the FP submissions by checking if they breach a curfew, in which case the FPs will be rejected. Otherwise, the NM requests their dissemination via the Radar.

IV. Messaging, events and interactions

The agent-based model architecture is static unless agents react to environment changes or requests triggered by interactions with other agents.

Simulation engine: Mercury is a discrete event-driven simulator. With this paradigm, agents react to events triggered by other agents or the environment. Once an event is resolved, the simulation jumps to the next event. Events can be rescheduled or cancelled, and new events can be created as required. A single queue of scheduled events is built for the whole simulator, with all the events stacked on a single timeline. In Mercury, events are linked to flight milestones: 'FP submission', 'Delay estimation', 'Passenger check', 'Push-back ready', 'Take-off', 'Flight crossing point', 'Landing', and 'Flight arrival'.

The reaction of one agent to an event might require the interaction of this agent with others in a message-driven approach, where agents react to messages sent by other agents, by the environment, or by a user.

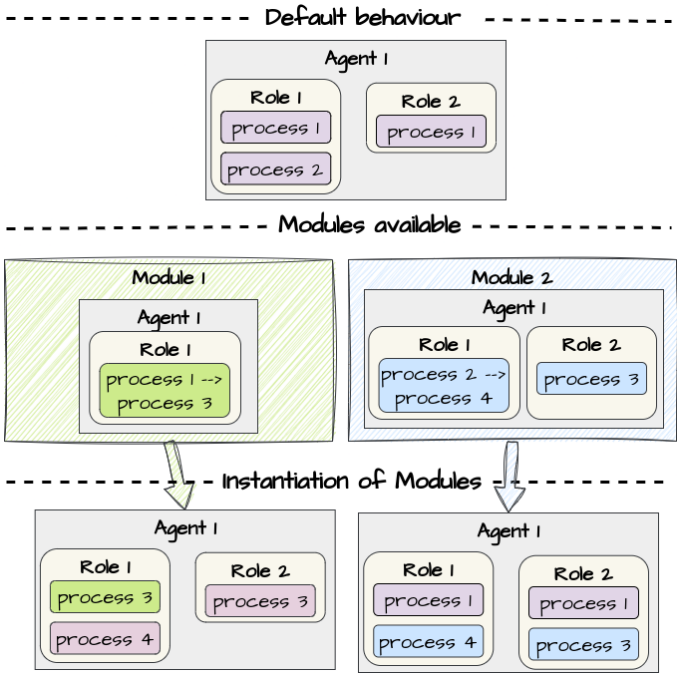

Figure 2. Example of Modules modifying the roles of an agent.

Some of the events in the simulation might be triggered at the same time (*e.g.* two flights with the same SOBT will have the same 'FP submission' time). Therefore, they are treated in parallel, leading to a case of concurrent programming. A system of queues and resources (using Python module Simpy) ensures these instances are properly managed.

Communication: Two types of communication exist: between agents (inter-agents) and within the roles of an agent (intra-agent).

V. Evaluation of ATM solutions

Mercury can evaluate different ATM Solutions by modifying the system's elements' behaviour. In this manner, Mercury becomes an evaluation platform for different ATM components. This can be achieved in two ways:

1) Replacing functionalities within agents' roles by the use of 'Modules'.
2) Following a microservice approach, connecting external systems to replace roles and/or agents from Mercury.

A. Modules

The functionalities of the roles within the agents can be modified when loading Mercury. This allows developers to create new functionalities indicating which roles should be replaced and added to the agents.

Figure 2 presents an example of the principles of this approach. A given agent (Agent 1) has two roles with some processes. This provides the implementation of the default behaviour of the agent. Two modules are available: Module 1 provides a different implementation for Process 1 of Role 1 (Process 3), and Module 2 provides a new implementation of Process 2 of Role 1 (Process 4) and an additional process to be added to Role 2 (Process 3).

For example, when a flight enters the scope of the arrival manager, a role within the E-AMAN requests the expected cost of using different available landing slots to the flight. With that information, the E-AMAN then assigns the slot for the flight. The first slot available could instead be provided. This completely different behaviour can be achieved by modifying selected roles within the E-AMAN with a Module.

As observed, if the Modules are instantiated, the roles are replaced and added as required into the agents. This flexible and versatile approach enables developers to create modules that modify and add specific functionalities to Mercury without modifying the base code. A set of modules can be loaded simultaneously to create a Case Study (*e.g.* a given modifying more than one agent at once). The developed code within the modules must be compatible with the remaining software, and developing these modules might require some deep knowledge of the inner workings of Mercury.

B. Microservices approach

Another approach that can be followed to evaluate external components with Mercury and change some components' behaviour relies on the microservices principles and exploits Mercury's message-based architecture for communication between agents (and roles).

Agents communicate in Mercury using an ad-hoc Delivery system. This system receives messages destined for agents and delivers them. This is achieved by adding the receiving agent's identifier in the message's header. It is, therefore, simple to modify this behaviour so that messages are serialised and communicated externally. This enables the broadcast of messages with a publisher-subscriber approach (*e.g.* to indicate the simulation status) and the establishment of a request/reply communication protocol (*i.e.*, one-to-one), miming the requests between agents. These types of architectures can be developed following some of the principles of microservices [16].

This approach allows for different technologies on the external elements to be used, albeit complexifying the communication protocols and the execution of a simulation. These external elements could replace agents or roles and be executed in the same or separate machine.

Mercury has already implemented and validated this concept by replacing some functionalities (selection of flight plans and prioritisation of flights in UDPP situations) by external actors with human-in-the-loop simulations [8].

A relatively complex example of interactions within Mercury elements consisting of two agents and four roles is presented in Figure 3. This example illustrates some of the characteristics of the communication channels within Mercury: intra and inter-agents, synchronous and asynchronous communications.

Process 1 of Role 1 in Agent 1 is waiting (asynchronously) for an event (e) to be triggered. When this happens, it requests some information from Process 1 of Role 3; as both roles

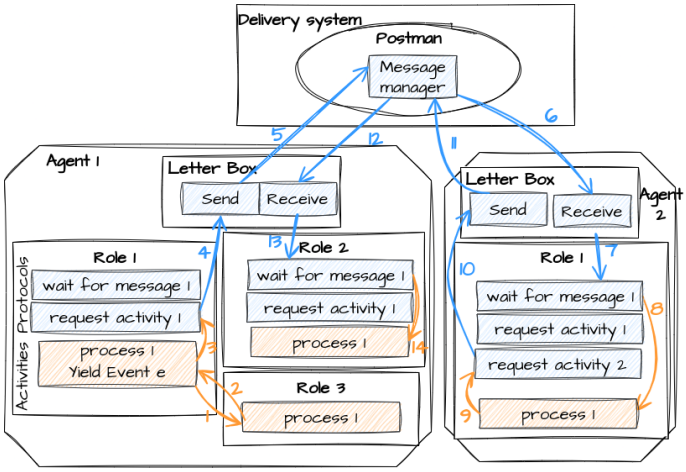

Figure 3. Example of communication between Agents and Roles in Mercury.

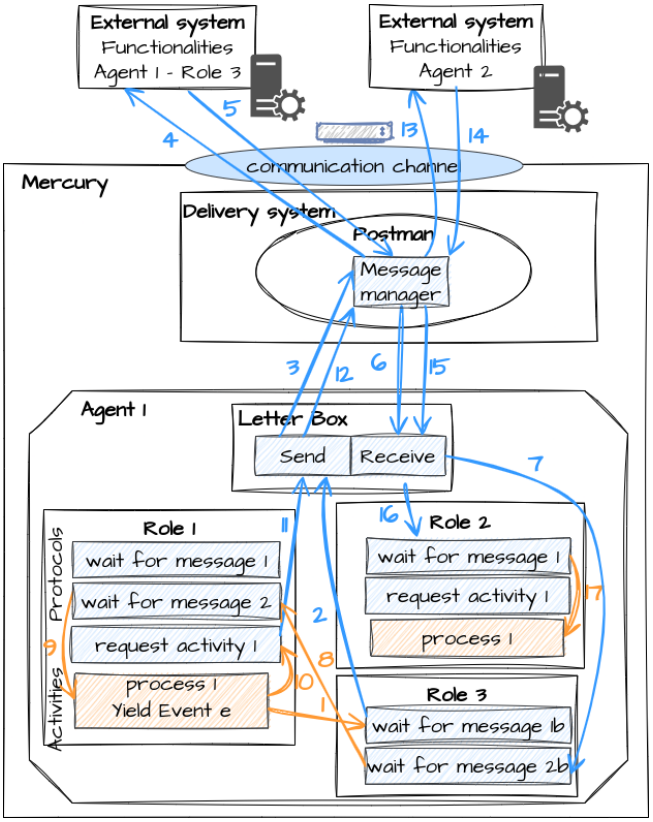

Figure 4. Same example of communication between Agents and Roles as in Figure 3 but with external elements replacing systems from Mercury.

are within the same agent, a direct interaction (intra-agent direct communication) is produced (1, 2). Process 1 then communicates with a service provided by Agent 2 (inter-agent communication). This triggers the request of an activity (3), which will send a message to Agent 2 via the Delivery system (4-5-6). Agent 2 identifies that the request should be processed by its Role 1 (7). This triggers Process 1, which, after some computing, requests an action to Agent 1 (9 to 12). In this case, the message from Agent 2 is treated by Role 2 within Agent 1 (13), who triggers the activities of Process 1 in that role (14). This process could, for instance, then yield until another event is satisfied.

Figure 4 presents the same interactions but with some elements executed as external systems: all services of Agent 2 and some functionalities of Role 3.

To replace Agent 2, the Delivery system redirects the messages from Agent 1 to the external communication channel (13-14). Note that communications between agents are generally already designed to support asynchronous interactions.

Two options are possible to replace Role3's functionalities from Agent 1. First, the inter-agent communication mechanism can be used, creating a message and sending it to the Delivery system, which will redirect it outside Mercury. Second, Role3 can manage the external communications maintaining the direct interaction between Role 1 and Role 3.

The second case is illustrated in Figure 4. Role 3 is modified to translate the request into an external message (1-3-4). Once a reply is obtained from the external system (5-6-7), Role 3 sends this back to Role 1 (8-9). This allows for some internal processes within a role to be externalised rather than the entirety of the role.

It is worth noticing that in both cases, the communication between Role 1 and Role 3 becomes asynchronous, which might need to be managed by Role 1. The simulation engine in Mercury also needs to avoid advancing on the events until the external system replies; the Delivery system ensures this. This approach successfully integrated human-in-the-loop simulations in the BEACON project [8] using Modules to replace the required processes within the externalised roles.

External entities might require information from the system's status beyond the data provided in the initial request. This means that a protocol of communication and message exchanges will be developed, providing an 'API' of Mercury that the external entities can query.

Finally, external systems should be able to treat each request from Mercury independent from the simulation status and/or manage internal information on the different Mercury runs being served by the system.

VI. Data and configuration

This section describes the input and output datasets and provides some information on the input and output managers (and their interface). All the datasets (except for configuration files) are stored in Apache parquet format in Mercury.

A. Configuration files

Configuration files in Mercury are defined using TOML format. These files configure some of the characteristics of Mercury and the simulations and define the modules' characteristics.

B. Input data

The data required as input are organised in three information levels:

1) Scenario: A Scenario represents all possible flights (with passenger itineraries and rotations) for a given period (usually a day) and a region (usually Europe). It also contains information required to simulate their operations in the ATM environment. Data is structured in ten categories: *schedules*, *passengers* (with passengers itineraries), *airports* (static and operational information), *airlines*, *aircraft performance*, *flight plans*, *delays* (probabilities primary delays), *eaman*, *costs* and *network* (ATFM regulations, probabilities and distributions).

For some of these data, note that not only a given value is necessarily provided, but different alternatives could be available. For example, the probability of being regulated by ATFM delay could be provided by different levels: 'High', 'Medium', and 'Low'; these could have been computed as representatives of historical days. These probability distributions are paramount for calibrating the model [17].

2) Case Study: A Scenario can have several Case Studies. These are defined by setting a configuration file that instantiates the parameters from the set of alternatives available in the Scenario — for example, setting the probability of ATFM delay to 'Medium'.

The Case Studies can also filter the data from the Scenario, particularly by reducing the set of simulated schedules, for example, using only the subset of flights operating at a given airport or departing/arriving in a given time window. Mercury will automatically read (and load) only the relevant information for the flights selected,*i.e.*, subset of passenger itineraries, airports, etc.

Additional data from the one in the Scenario might need to be stored, *e.g.* a list of flights to be simulated. A Case Study can also override some parameters from the Scenario, *e.g.* the radius of the planning horizon of the E-AMAN at a given airport. All these additional data will be saved within the Case Study.

Finally, a Case Study can define which Modules from Mercury should be applied.

3) Experiment: We might want to explore the impact of some parameters on the results; this will require the definition of experiments. For example, indicating a set of values of fuel cost to evaluate. These parameters will be defined in the experiments' TOML configuration files.

Finally, as Mercury is a stochastic simulator, operational parameters, such as en-route flight route deviation, turnaround time or amount of non-ATFM delay, are drawn from predefined probability distributions, which need to be calibrated with historical datasets [17]. Therefore, several simulator runs might be required to obtain statistically significant results. These configurations of the execution of Mercury, along with information on computational settings, *e.g.* the number of parallel executions, will be defined in the experiment and the simulation configuration files.

C. Output data

The output data keeps the traceability with the input used to generate them. Therefore, the output data are structured similarly to the input while keeping a copy of the configuration files used to run the different experiments.

Finally, Mercury stores the different runs and the aggregated KPIs computed using the information from all the runs of a given experiment to obtain statistically significant values. These aggregations are computed by a Result Aggregator, which can be configured as part of the experiment definition.

The output can be post-processed to compute more advanced metrics such as door-to-door mobility (by adding access and egress times to the individual passenger itineraries metrics) [10], network complexity parameters [18] or analysing the emergent agents' behaviour [17].

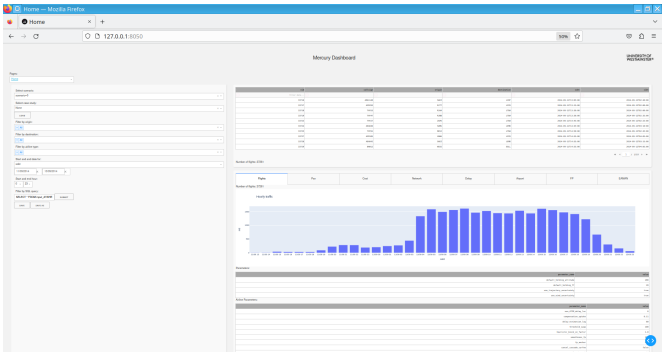

Figure 5. Web-based user interface.

D. Computational time

The computational requirements continue to be contained, with a simulation of a whole day of operations in Europe[12] requiring around 9Gb of RAM and 30 minutes computation (on a CPU@3.2GHz). Moreover, as presented in [17], the resources per individual flight decrease as the number of flights increases, showing good scalability.

E. User interface

Two dedicated modules are developed as part of Mercury to facilitate the manipulation of the input and output datasets: Input manager and Output manager.

A web-based graphical interface which connects to these managers has been developed, providing basic functionalities to explore the input and output data and create the Case Studies, Experiments and the configuration of Mercury.

Figure 5 presents this interface where the flight information in the selected scenario is presented as a table and a distribution of departing times.

The user can filter the flights available in the Scenario by origin, destination, airline type, start and end date of their schedules or even directly using an SQL query that will be applied to the flight schedule table. This filtering of flights is the first step towards creating a new Case Study. These Case Studies can be loaded, if available, within a given Scenario.

[12] Approximately 27k Flights (with 7k individual aircraft), 300 airline operating centres, 800 airports and 400k passenger groups (representing 3.4M passengers)

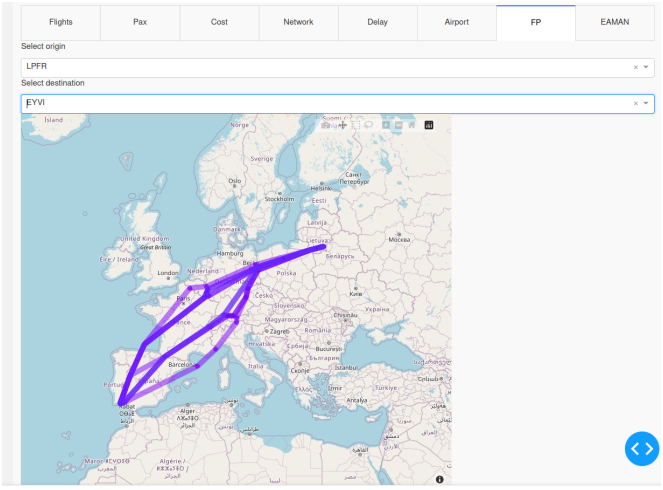

Figure 6. Flight plan alternatives exploration interface.

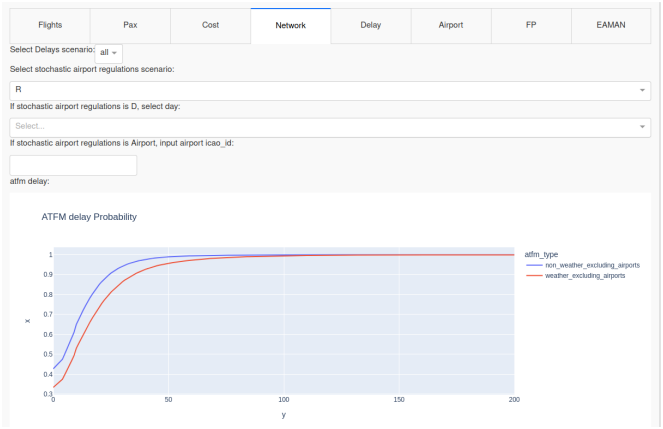

Figure 7. Network-ATFM data visualisation and selection.

The different data groups presented previously can then be explored for the subset (or the entirety) of flights using the available tabs. For example, Figure 6 shows the visualisation of the flight plan pool filtered by a given origin and destination. In some cases, as in Figure 7, the user can visualise the data available in the Scenario (*e.g.* distribution of ATFM delay) and also select which parameters should be used (*e.g.* which airports should have ATFM regulations in this Case Study).

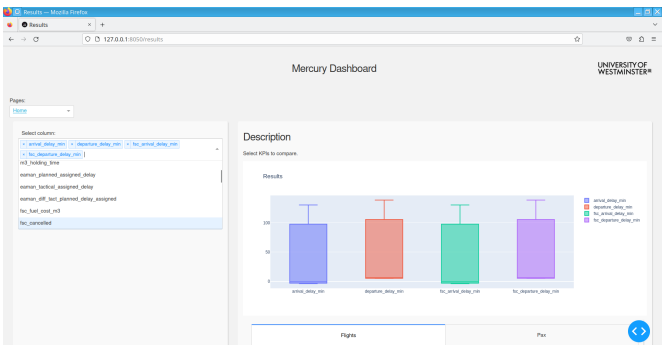

Figure 8. Results analysis and visualisation.

Finally, the current interface version allows users to explore the results of the execution of Mercury by processing the required output folders, as shown in Figure 8. Note how not only average results per KPI are produced, but percentiles and distribution of these can be estimated.

VII. Conclusions and Future Work

Mercury has proven its capabilities to integrate new behaviours and mechanisms for their evaluation, considering network effects, complex interactions, plus flight and passenger metrics. The current architecture is flexible, and using modules to modify individual roles within the agents allows user-friendly modification of such behaviours. Moreover, in recent projects, effort has been devoted to improving the ease of integrating external modules and solutions (*e.g.* supporting human-in-the-loop simulations with interaction with an external interface [8]). These efforts will continue towards developing a full messaging API enabling the interaction of external systems with simulations within Mercury. This will be done particularly in the context of air-rail multimodality by supporting the evaluation of tactical multimodal disruption management solutions [4].

Future work will aim at standardising the usage of modules and external systems integration facilitating the wider usage of Mercury. The interface for manipulating input and output datasets will also be expanded to facilitate the creation of case studies and experiments. The low-level resolution of Mercury enables the computation of many advanced metrics as post-processing. This will also be further developed and provided as part of the interface system. The datasets (input/output) must be fully documented, providing the required meta-data to develop new scenarios.

Opening Mercury will allow it to evolve with future needs for performance assessment from the wider research community. We hope that it will foster a new era of open models in air transportation that can be used to reproduce results from different research groups, to quickly prototype new modifications of the system, to interface local models with network ones to assess the impact of solutions at a higher level, and finally to push for the standardisation and modularity of air transportation simulators. We also hope this will help develop standard datasets that can be used routinely to evaluate solutions in different simulators.

We invite any researcher interested in assessing new processes and technologies to try the simulator and contact the authors.

References

[1] European Commission, “Regulation (EC) No 261/2004 of the European Parliament and of the Council of 11 February 2004 establishing common rules on compensation and assistance to passengers in the event of denied boarding and of cancellation or long delay of flights, and repealing Regulation (EEC) No 295/91,” pp. 1–7, 2004.

[2] A. Cook, G. Tanner, S. Cristóbal, and M. Zanin, “Passenger-oriented enhanced metrics,” in 2nd *SESAR Innovation Days*, 2012.

[3] High Level Group on Aviation Research, “Flightpath 2050 – Europe’s Vision for Aviation,” European Commission, Tech. Rep., 2011.

[4] L. Delgado, T. Bolić, A. Cook, E. Zareian, E. Gregori, and A. Paul, "Modelling passengers in air-rail multimodality," in *Proceedings of the 11th EUROSIM Congress*, July 2023.
[5] A. Cook, L. Delgado, G. Tanner, and S. Cristóbal, "Measuring the cost of resilience," *Journal of Air Transport Management*, vol. 56, pp. 38 – 47, 2016, long-term and Innovative Research in ATM. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S096969971600020X
[6] U. Kluge, A. Paul, H. Ureta, and K. O. Ploetner, "Profiling future air transport passengers in Europe," in *Transport Research Arena (TRA) 2018*, 2018.
[7] S. Zaoli, P. Mazzarisi, F. Lillo, L. Delgado, and G. Gurtner, "New centrality and causality metrics assessing air traffic network interactions," *Journal of Air Transport Management*, vol. 85, 2020.
[8] BEACON Consortium, "Deliverable D5.2: Final tactical model and results," BEACON Consortium, Tech. Rep., February 2023. [Online]. Available: https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5f3a1058d&appId=PPGMS
[9] J. Hoekstra and J. Ellerbroek, "BlueSky ATC simulator project: an open data and open source approach," in *7th International Conference on Research in Air Transportation (ICRAT)*, 06 2016.
[10] L. Delgado, G. Gurtner, A. Cook, J. Martín, and S. Cristóbal, "A multi-layer model for long-term KPI alignment forecasts for the air transportation system," *Journal of Air Transport Management*, vol. 89, p. 101905, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0969699720304889
[11] G. Gurtner, C. Bongiorno, M. Ducci, and S. Miccichè, "An empirically grounded agent based simulator for the air traffic management in the SESAR scenario," *Journal of Air Transport Management*, vol. 59, pp. 26–43, 2017.
[12] G. Gurtner and T. Bolić, "Impact of cost approximation on the efficiency of collaborative regulation resolution mechanisms," *Journal of Air Transport Management*, vol. 113, p. 102471, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S096969972300114X
[13] A. Nuic, D. Poles, and V. Mouillet, "Bada: An advanced aircraft performance model for present and future atm systems," *International Journal of Adaptive Control and Signal Processing*, vol. 24, no. 10, pp. 850–866, 2010. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/acs.1176
[14] M. Wooldridge, N. R. Jennings, and D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design," *Autonomous Agents and Multi-Agent Systems*, vol. 3, pp. 285–312, 2000.
[15] Domino Consortium, "Deliverable D4.1: Initial model design," Domino Consortium, Tech. Rep., November 2018. [Online]. Available: https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5bf17614a&appId=PPGMS
[16] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, *Microservices: Yesterday, Today, and Tomorrow*. Cham: Springer International Publishing, 2017, pp. 195–216. [Online]. Available: https://doi.org/10.1007/978-3-319-67425-4_12
[17] G. Gurtner, L. Delgado, and D. Valput, "An agent-based model for air transportation to capture network effects in assessing delay management mechanisms," *Transportation Research Part C: Emerging Technologies*, vol. 133, p. 103358, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X21003600
[18] S. Zaoli, P. Mazzarisi, and F. Lillo, "Trip centrality: walking on a temporal multiplex with non-instantaneous link travel time," *Scientific Reports*, vol. 9, no. 10570, 2019.