Issue 26(1), 2026
ISSN: 1567-7141
Special Issue: Sesar Innovation Days 2023
DOI: 10.59490/ejtir.2026.26.1.7529

TU Delft OPEN Publishing
Research Article

Mercury: An open-source simulator for the evaluation of air transport mobility

Luis Delgado [1], Gérald Gurtner [1,*], Michal Weiszer [1], Tatjana Bolić [1], Andrew Cook [1]

[1] Centre for ATM Research, University of Westminster, United Kingdom
* corresponding author g.gurtner@westminster.ac.uk

Keywords
air transportation
agent-based model
performance assessment
simulator
open-source
passengers
airline cost model
simulation as a service

Publishing history
Submitted: 26 April 2024
Revised date: 03 February 2025
Accepted: 28 February 2025
Published: 31 March 2026

Cite as
Delgado, L., Gurtner, G., Weiszer, M., Bolić, T., & Cook, A. (2026). Mercury: An open-source simulator for the evaluation of air transport mobility. *European Journal of Transport and Infrastructure Research, 26*(1). https://doi.org/10.59490/ejtir.2026.26.1.7529

Abstract

The Mercury simulator, a performance assessment platform, is a stochastic, agent-based model developed over several years during research projects. It features a detailed description of the air transportation system at the European level, including passengers and aircraft, and various important actors such as the Network Manager, airports, etc.

This article presents the possibilities offered by the simulator's current, now open-source version. We describe the core Mercury functionalities and highlight its modularity and the possibility of its usage with other tools. We present a new interface, which supports user-friendly interaction with the simulator, exploring data input/output and parameter settings. We emphasise possible uses as a solution performance assessment tool, which is usable early in the innovation pipeline to better estimate the impact of changes and new systems in the air transportation system. We hope that opening the simulator may encourage other developers to open their models, allowing faster prototyping of new operational concepts early in the innovation pipeline and an *in fine* support standardisation and higher performance of simulation-based performance assessment tools.

1 Introduction

Air transport is a key component of the European economy, and effective and safe air traffic management (ATM) is its essential component. The Single European Sky Air Traffic Management Research (SESAR) programme aims to modernize and enhance Europe's ATM system. The outcomes of the SESAR programme are the ATM Solutions - a Solution represents a change in the way ATM is performed, they are new operational concepts, procedures, and/or relevant technologies. Often, these Solutions are assessed on a subset of the air transport system, making it difficult to evaluate their network-level impact, especially when considering the interactions with other innovations introduced to the system. Appropriate performance assessment, at network level, needs to be performed for further deployment decisions, or testing of the impact of novel transport policies on the system. Mercury is a simulator that is developed to serve as a performance assessment platform, be it for the impact of Solutions or policies.

Further, the air transport system performance is driven by uncertainty, disruptions, and the interaction of many different actors. By their nature, these actors tend to make decisions in sub-optimal conditions: reacting to disruptions, estimating uncertainties in the system without the full knowledge of the other actors' intentions. Mercury's 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. Even with simple rules, emergent behaviours arise from the interaction between agents in an evolving environment, capturing the richness of the network interactions in the system.

Passengers represent the important mobility stakeholders. Usually, envisioned improvements are designed to improve one or the other facet of passenger experience, from punctuality/delay to quality of service. Thus, it is important to be able to assess the system performance also from the passenger's point of view, which is enabled in Mercury. Previous research (Barnhart et al., 2014; Bratu & Barnhart, 2005; Cook et al., 2012) has proven that passenger delays need to be modelled to assess the system's performance, besides the 'nominal' key performance indicators (KPIs) used by the air traffic management (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 (European Commission, 2004)) and therefore drives some of the behaviour of airlines; and second, as passengers experience delays that are different from flight delays, due to connections (Cook et al., 2012) and the fact that they need to reach their final destination in a door-to-door journey (European Commission: Directorate-General for Research and Innovation & Directorate-General for Mobility and Transport, 2011), which might include multi-modal trips (Delgado et al., 2023). *Therefore, there is a need for a simulator able to evaluate ATM solutions and policies flexibly (modifying agents' behaviour or assessing external systems), taking into account passenger-related considerations.*

The Mercury simulator has been developed in this context for over ten years. Through different projects, a first simulator version able to capture passenger-centric indicators (Cook et al., 2012) was expanded to consider cost resilience (Cook et al., 2016) and door-to-door journeys (Kluge et al., 2018). The simulator was redesigned as a full agent-based model, enabling the development of new metrics to capture network effects (Mazzarisi et al., 2020). 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 an external interface (Gurtner & Bolic, 2023a), or adding rail-related mobility models to the simulated network). Mercury incorporates the processes and behaviours of actors in the environment, considering their expected cost functions. Because any change in regulation (like the changes to passenger compensation through EU Regulation 261 (European Commission, 2004)) modifies the costs and revenues of the actors (or their utility), Mercury provides not only a technological and procedural evaluation platform but also the possibility for policy evaluation.

Other available simulators tend to be flight-centric and lack cost and passenger modelling. This makes it difficult to properly model the actors' decisions, and low-level KPI distributions cannot be extracted as in Mercury. However, they can provide higher performance on the evaluation of particular flight-centric aspects of ATM. For example, EUROCONTROL's R-NEST provides highly detailed airspace and capacity management modelling (EUROCONTROL, n.d.-b), while the open simulator BlueSky (Hoekstra & Ellerbroek, 2016) provides detailed trajectory integration in time-based simulations of flights. Neither of these detailed modelling of airspace or flight trajectory is captured to such detail in Mercury, as the focus is on passenger flows.

In contrast with other available tools, Mercury captures the *emergence of ATM performance* from the behaviour of different agents, which react based on expected operational costs and individually defined rules. This enables the *modelling of the decision-making processes of key actors* (particularly airlines)[1] and not only the evaluation of Solutions but also policy changes (as they would translate into different expected costs and hence decisions when managing flights) and operational procedures defined by the actors. 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[2] region in a few minutes. This kind of speed is required to produce several runs needed to obtain statistically significant results, as Mercury relies on stochastic processes which describe different aspects of the ATM processes (see Annex 0).

Over the years, Mercury has been used in various research projects for different objectives. In particular, it was used for the following SESAR Exploratory Research projects:

- POEM (Cook et al., 2012): MatLab implementation, used for computation of reactionary and passenger delays.
- ComplexityCost (Cook et al., 2016): Extension to previous Matlab implementation, used for computation of tactical costs of ATM mechanisms to estimate the cost of resilience under different disruption scenarios.
- Vista (CORDIS, 2022c; Delgado et al., 2020): first Python implementation, functional, aiming at estimating reactionary and passenger delays, and various other KPIs, including cost of delay and cost of fuel.
- Domino (CORDIS, 2025; Gurtner et al., 2021; Mazzarisi et al., 2020): new Python implementation, object- and agent-oriented, aiming at allowing more flexible rules for airspace users.
- BEACON (CORDIS, 2023a Gurtner & Bolić, 2023b): and NOSTROMO (CORDIS, 2023b; Riis, et al., 2024): the second version of the new implementation, with extensions to support:
    - complex behavioural rules for agents, User-Driven Prioritisation Process solutions, interface for human-in-the-loop simulations (for BEACON).
    - easier input/output management for metamodelling purposes, extended arrival manager solutions (for NOSTROMO).
- MultiModX (CORDIS, 2023b, Delgado et al., 2023) (in progress, see Section 8.2): the third version of the new implementation, revamped for open uses, added support for third-party optimisers and plug-ins. Added agents and processes for landside integration, i.e., ground mobility and air-rail exchanges. Added complex logic for airport passenger management processes.

Mercury has also been used in other research and innovation actions, including SESAR DATASET2050 (CORDIS, 2022b), CAMERA (CORDIS, 2022a), etc.

---

[1] Note that we consider airlines as key actors but not passengers, because passengers are mostly passive once they have their tickets, i.e., they will follow the processes chosen by the airline. The latter makes all the important decisions for them, in particular managing their flights, as well as their rebooking options in case of disruption.
[2] ECAC stands for European Civil Aviation Conference and encompasses 44 Member States.

In summary, Mercury has been used in one form or another to estimate the following:

- impact of changes in processes in the air transportation system,
- impact of macroeconomic parameters (e.g., price of fuel),
- impact of new technologies,
- impact of network effects,
- impact of behavioural changes (mainly airlines).

All estimated impacts are of tactical nature. Thanks to the wide range of indicators computed by the model, the following types of impact can be measured:

- overall impact on airlines regarding costs, delays, etc.
- impact on passengers in terms of real arrival times, disutility, etc.
- environmental impacts.
- differential impacts (different types of passengers, airlines, airports, etc).
- trade-off impact (for instance, CO_2 vs costs).

Given the need for European-wide assessment tools that support multi-KPI estimations on the one hand, and the capabilities of Mercury on the other, it was decided to open the code source of Mercury, making it available to the wider research community. Opening Mercury source code is aligned with and a part of the Open Science Alliance for ATM initiative (Bolić et al., 2023), which sponsors the adoption of open science principles in ATM research. Mercury is distributed under a GPL v3 licence, which implies that any distribution, in entirety or of part of its code, has to be done under the same terms. *The code is available on GitHub*[3], and details on the software characteristics can be found in Annex 0. 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 true air transport mobility performance analyser.

This article briefly describes the workings of the simulator, its main strengths, how it can be used for new assessments – for instance, developing new modules/add-ons, or interfacing with existing models – and the type of interface available to set up simulations and explore 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 further development, rather than presenting the experimental results of running the simulator in various projects.

The paper is structured as follows: Section 2 introduces the specification and design of the model. The agents are described in Section 3 while Section 4 highlights Mercury's simulation engine and communication mechanisms. Section 5 shows how new mechanisms and behaviours can be implemented for their evaluation with the platform and how external systems can be connected. The data used (and produced) by Mercury is described in Section 6. Aspects relating to computational performance and user interface are described in Section 7. The paper closes with current and future planned developments in Section 8 and conclusions in Section 9. Two annexes complement the article: Annex 0 present the probability distributions required and modelled in Mercury and Annex 0 provides more details on software implementation aspects.

2 Model specification and design

The Mercury's agents and their interaction emerge from the agent-based modelling approach followed in the model design.

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

2.1 Development methodology

The Gaia methodology has been used to build the model (Wooldridge et al., 2000). Gaia is a methodology for agent-oriented analysis and design applicable to multi-agent systems based on the view that a multi-agent system is a computational organisation consisting of various interacting roles.

The methodology divides the modelling activities into two phases: analysis and design. First, an analysis is conducted to understand the system (roles and interactions).

1. Roles: A role can be seen as an abstract description of an entity's expected function, which is defined by four attributes:
    a. Responsibilities: determining the functionality of the role,
    b. Permissions: rights associated with a role, describing the resources available to the role (e.g., reading, modifying, or creating data),
    c. Activities: computations associated with the role that can be carried out without interacting with other roles,
    d. Protocols: these define how a role can interact with other roles.
2. Interaction: Roles have limited access to information and cannot access private information of other roles. Hence, they need to interact, i.e., delegate computations (and request information) to other roles. The interaction model lists all these possible interactions between roles.
   In the design phase, the roles are bundled to create agents that provide services.
3. Agents: Once all the roles and their interactions are defined, they are grouped into higher-level entities representing different agent types. This process is generally largely arbitrary and adaptable as the specification of the system evolves into a particular designed architecture.
4. Services and acquaintances models: As the agents are defined as collections of roles, they inherit their protocols and activities: the roles' activities become agents' activities, and roles' protocols are turned into agents' services. Protocols can trigger interactions between roles that are part of the same agent or between roles belonging to different agents. In the first case, the interaction can be reduced to a direct request, given that both roles share the same memory space within the agent. In the second case, a message is sent between agents requesting some computation (or data) using non-shared resources. This can be viewed as a 'service' provided by one agent to the other.

The service list of each agent can be understood as that agent's Application Programming Interface (API) or its public interface. The standardisation of these interactions increases the reusability and flexibility of the model.

Grouping roles to agents creates 'acquaintances', i.e., required visibility between agents. This provides a high-level view of the model, which is useful to estimate the degree to which different parts of the modelled system are coupled, i.e., are interdependent on each other (see Figure 1 for diagram).

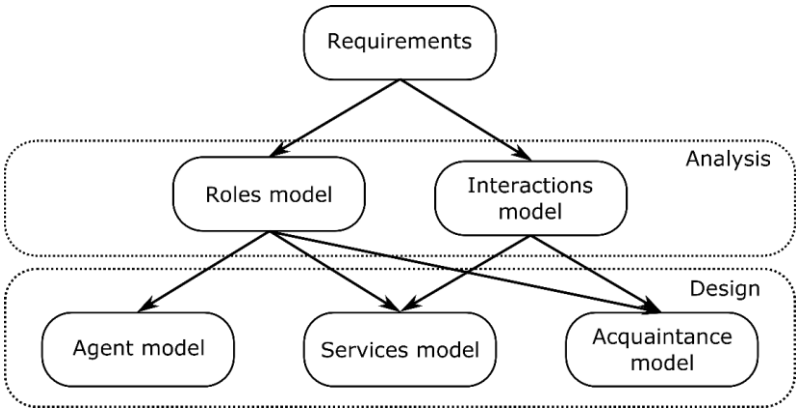

Figure 1. Flow of design in Gaia's model (from Wooldridge et al. (2000)).

2.2 Application of methodology in the context of Mercury

The first step of the methodology is to define the scope and requirements for the model. Mercury aims to capture the interaction between the elements of the ATM system to provide performance indicators for key stakeholders with a focus on evaluating a day of operations, including pre-departure processes, tactical operations of flights, and passengers' itineraries. The model can:

- 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 facilitates the change 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, focusing on reactionary delay and passengers' itineraries.
- Consider fairly complex decision-making processes from the airlines when reacting to cost and (not only) delay.

The model considers the most important channels of propagation of delay. First, individual aircraft are used throughout the day for different, consecutive flights, and thus delays can propagate (i.e., reactionary delay). Second, passengers are modelled to capture the costs of their delays and missed connections. Third, the model considers exogenous sources of delays, e.g., turnaround delays. Fourth, the system's reactions to delays and/or congestion are included. Finally, the model allows dynamic decisions from agents.

Mercury does not capture safety aspects per se. These are considered only indirectly through defining capacities for different subsystems (e.g., airports' runway/s throughput) and modelling the consequences of enforcing them (i.e., generating delays).

With the previous considerations, forty roles are identified in Mercury as listed in Table 1. The reader is referred to (Delgado et al., 2018) for a detailed description of these. Some aspects of the ATM system are kept at a high-level, this is particularly the case for the en-route tactical operations (e.g., flight conflict detection and resolution). The description level has been chosen to strike a good balance between computational complexity and microscopic description of the model. How the elements of the system react to requests and the environment can be modified, e.g., to test a new solution, only at the same or larger scale as the rules and elements described in the model. In other words, only modifications of the inner decisions of the agents (activities/responsibilities) or their interactions (protocols) can be done with the model without requiring a disaggregation of these, as explained in Section 5.

Table 1. Agents and their roles.

Agent type	Roles
Airline Operating Centre	Airline Flight Planner, Dynamic Cost Index Computer, Passenger Reallocation, Turnaround Operations, Airline Passenger Handler, Flight Plan Selector
Flight	Aircraft Departing Handler, Departure Slot Requester, Flight Plan Constraint Updater, Flight Plan Updater, Flight Arrival Information Provider, Ground Arrival Handler, Operate Trajectory, Potential Delay Recovery Provider
Ground Airport	Ground Handler, Provide Connecting Times, Taxi-out Estimator, Taxi-Out Provider, Taxi-In Provider
E-AMAN	Strategic Arrival Queue Builder, Arrival Queue Planned Updater, Arrival Cancellation Handler, Flight In AMAN Handler, Arrival Planner Provider, Arrival Tactical Provider, Slot Assigner, Arrival Planner Provider Queue, Arrival Tactical Provider Queue
DMAN	Strategic Departure Queue Builder, Departure Slot Provider, Departure Queue Updater, Departure Cancellation Handler
Radar	Disseminate Flight Plan, Disseminate Cancellation FP, Disseminate Flight Position Update, Radar Augment Flight Plan
Network Manager	Network Manager Flight Plan Processing, Network Manager Accept and Disseminate FP, Network Manager Cancel FP

Initially, sixty-three types of interactions between roles were identified in the model — see (Delgato et al., 2018) for a full description.

The process to bundle the roles into agents was guided by existing entities in the ATM domain, with naming and representation referring to a coherent actor/entity, such as the Airline Operating Centre (AOC). This facilitates the identification of agent types with well-known entities in the ATM community containing well-defined functionalities (roles). Another important consideration when defining the agent types is the private nature of the information available within an agent. An agent is the 'atomic level', above which information is not freely accessible. Within an agent, i.e., within and between its roles, all information can be shared. This does imply repercussions in terms of performance (e.g., time required to access and process information) and behaviour (e.g., information available when performing decisions).

Following these guidelines, seven agent types are defined, as indicated in Table 1. The inner workings of the model and details on the agents' decision-making process are presented in Section 3.

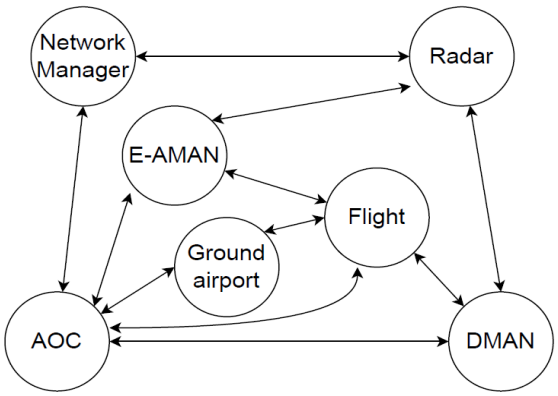

Figure 2. Acquaintances between agent types, derived from interactions between their underlying roles.

Finally, from the interactions between roles, the services provided by agents and their acquittance emerge, as shown in Figure 2, where it is easy to observe the centrality of the AOC agent. Section 4 provides more details on how the communication between agents is modelled in Mercury.

3 Agent's description

In this section, we briefly describe all the agents present in the platform, mentioning their main services, how they react to events, and what their inner logic is.

3.1 Airline Operating Centre (AOC)

Mercury has one 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[4]).
- 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 (European Commission, 2004), and 'soft' costs, of a future loss due to passengers' dissatisfaction and the subsequent impact on the market share based on (Cook & Tanner, 2015). 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 when considering this.

The crew could be incorporated in Mercury following a similar approach to the modelling of passengers, as explained in the next point below. However, the crew management and its associated explicit costs due to disruption are not modelled due to the limited data available for these computations. The associated costs are aggregated as part of the cost of delay function based on (Cook & Tanner, 2015). A similar approach is used for other costs, such as maintenance.

Finally, note that the AOC uses different cost estimation functions depending on the phase of the flight considered. Some costs relate to departure delay (e.g., duty of care) while others are impacted by arrival delay (e.g., reactionary delay). Therefore, in some cases, when deciding to speed up a flight after departure, some costs might already have accrued and should not be considered. In all cases, delays are computed with respect to the schedules and the times at the gate; hence, buffers and padding in the schedules play an important role.

Modelling passengers

Passenger groups are modelled as simple placeholders which contain information on the passenger itineraries and their characteristics (flight sequence, type of passengers ('standard', 'flex'), fare, and number of passengers in itinerary group), which other agents, notably the AOC, handle. Passenger groups are 'attached' to flights and airports.

Response to events

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

[4] Modifying the flight plan using changes in the speed and/or vertical profile to trade delay and fuel.

- '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 based on their expected costs (including en-route charges, expected fuel usage, and delay costs). 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 reconsideration of which FP to use. The NM could reject a flight plan if it would breach a curfew, and the AOC could also decide to cancel the flight.
- '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 that will be missed, a new slot is requested from the NM. If the delay exceeds 30 minutes, a new FP is recomputed.
- 'Passenger check': 5 minutes before EOBT, the AOC computes the expected arrival time of connecting passengers who are supposed to board this flight but are not there yet and decides whether to wait.
- 'Push-back': At the push-back event, the AOC checks which passengers are on board and computes the passengers' metrics such as departure 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, seats available, fares, and ultimately, assigning Reg. 261 compensation, as required. At push-back, the AOC might also estimate if the FP should be adjusted by modifying the planned trajectory and communicating this to the Flight agent.
- '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.

Relevant services

The AOC interacts with other agents, some of the more relevant services are:

- passenger processing: *allocation passenger request, process arrival passengers request*,
- assess and generate flight plans: *departing reassessment request, request flight plan, select flight plan option request, hotspot*[5] *decision request*,
- react to flight status update: *notification of top of climb reached, cost function estimation request.*

3.2 Flight

'Flight' agents integrate flight movements (ground and trajectory). They also capture the actions performed by the crew e.g., requesting a departure 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, like fuel usage.

[5] Referring to ATFM regulation prioritisations.

- '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 time-window requests, like to land within a slot provided by the E-AMAN.

Relevant services

The services provided by the Flight enable the:

- update of flight plan (pre-departure): *flight plan update*,
- adjustment of trajectory following AOC request: *speed update,*
- provide information to other systems on the flight and estimated operations, particularly to the E-AMAN: *arrival information request, estimated landing time request, potential delay recovery request*,
- adding constraints to the flight trajectory to meet time-window requirements, like landing at a slot requested by the E-AMAN. These will be implemented by speed adjustment and/or holding at arrival: *controlled landing time constraint request, departure slot allocation*.

3.3 Ground Airport and its services

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 operations such as airport, airline, aircraft, and passenger types through pre-computed probability distributions (more information about these distributions can be found in Annex 0). One Ground Airport agent is instantiated per airport.

The Ground Airport provides services (relevant services) required to:

- compute turnaround times: *turnaround time request*,
- compute passengers' connecting times: *connecting time request*,
- taxi-in and taxi-out times estimation and computation: *taxi-out estimation request, taxi-out time request, taxi-in time request.*

3.4 DMAN and E-AMAN and their services

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 airport capacity information, and they might be adjusted if ATFM regulations are issued at the airport. All airports have an E-AMAN to manage the arrivals; for some, a planning horizon for trajectory adjustment and reduction of holding time is also implemented.

These services (relevant services) are required to manage the queues of slots at the runway:

- managing requests for departure slots and releasing slots due to cancellations: *departure request, flight plan cancellation*,
- being notified of flight entering the E-AMAN: *flight arrival information, flight in execution horizon.*

3.5 Radar

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 Points` considering the registered request for notification.

Relevant services

To perform these activities, the Radar provides these services:

- subscribe agents to being notified when flights reach certain positions: *subscription request*,
- augment flight plan adding waypoints: *flight plan augmentation request,*
- disseminate flight plan information: *flight plan dissemination request, flight plan cancellation dissemination request.*

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.

3.6 Network Manager (NM) and its services

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 explicit knowledge of the airspace. Instead, the NM uses:

- random en-route ATFM delays, based on empirical data from the analysis of historical EUROCONTROL's DDR2 datasets (EUROCONTROL, n.d.-a) from which the probability of being regulated is calculated, and if the flight is regulated, the distribution of the amount of delay issued is estimated (see Annex 0),
- regulations at airports modelled as queues 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.

The services (relevant services) provided to manage the FP submission and regulations management:

- FP submission and ATFM delay management: *flight plan submission, ATFM request, flight plan cancellation*.

4 Messaging, events, and interactions

An agent-based model architecture relies on the presence of agents interacting autonomously with each other and with the 'environment' - all processes not strictly included in agents (like ATFM regulation handling). The architecture is static unless agents react to changes in the environment or requests triggered by interactions with other agents (or actors). The Gaia methodology does not prescribe a specific implementation approach. Therefore, there is no imposition on the simulator engine, nor on how the communication channels between the agents should be implemented. Different alternatives were considered, as detailed in (Delgado et al., 2018), before settling on the current implementation.

4.1 Simulation engine

Mercury is an 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 in the queue. 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 system, with all the events stacked on a single timeline.

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.

In the usual setting, roles will be waiting for the trigger of an event; when that happens, a set of actions will be performed (activities), which might require the interaction with other roles (within the same agent) or the request for information or delegation of computation to other agents (services). Note that these requests for information tend to be asynchronous; therefore, waiting for replies may be required to implement. Once all the activities (including the interaction between agents) triggered by the event are completed, the simulation jumps to the next event in the queue. The agents' actions might modify the scheduled events, and in some cases, internal events for synchronisation might also be created.

Some of the events in the simulation might be triggered at the same time (e.g., two flights with the same scheduled off-block time (SOBT) will have the same 'FP submission' time). Therefore, they are treated in parallel, leading to concurrent programming. A system of queues and resources (using the Python module Simpy) ensures these condition races are properly solved[6]. A particularly relevant resource is the aircraft, which ensures that a flight is not operated unless the previous flight and turnaround processes are finished, enabling the explicit modelling of reactionary delay.

The asynchronicity of Mercury is managed by the process-based discrete-event simulation framework Simpy (SimPy, 2023), which provides an asynchronous event dispatcher. This library provides an event-loop manager for discrete-event simulations with progression of simulated time, skipping idle periods, and therefore providing fast-time simulation.

In Mercury, events are linked to flight milestones. Table 2 shows all the events in the model.

Table 2. Events used in the simulation.

Event	Short description
FP submission	First submission of the flight plan for a flight. 3 hours before the flight's SOBT.
Delay estimation	AOC checks the status of the flight, and a random non-ATFM delay is drawn one hour before the flight's EOBT.
Passenger check	AOC checks which passengers are not ready to board their flights, 5 minutes before EOBT.
Push-back ready	The aircraft is ready to push-back. The flight requests a departure slot.
Push-back	The flight is off-block and begins taxi-out. Connecting passengers who are not boarded are rebooked.
Take-off	The flight begins an "operate trajectory" activity, which integrates the trajectory between pre-defined waypoints in the flight plan.
Flight Crossing Point	A waypoint is crossed by the flight during its trajectory execution. The event triggered by the flight and captured by the Radar, which broadcasts the position of the flight to interested parties.
Landing	The flight reaches its final trajectory point, begins taxi-in.
Flight arrival	The flight arrives at the gate, turnaround and connecting passenger processes begin.

[6] Condition races happen when two different processes try to access or modify a common resource (for example the EOBT) during the same event, for instance, a service in the Flight agent and one in the Network Manager. Note that this is independent of multiprocessing or multithreading and happens even with single-threaded computations.

4.2 Communication

Two types of communications can be differentiated: messages between agents (inter-agent communications) and requests within roles in a given agent (intra-agent communications).

Inter-agents communication

The number of messages exchanged between agents can be very large. For a day of operations in Europe, Mercury generates close to 1.4M messages. This is for approximately 27k Flights, 300 AOCs, 800 Ground airports, E-AMAN and DMAN instances, and around 400k passenger groups (representing around 3.4M passengers).

Therefore, there is a need for a high-performance communication approach. An ad-hoc messaging system is used. Every agent has a 'Letter Box', which allows the reception of standardised 'Letters' being delivered to the appropriate recipient using a centralised 'Postman'. Agents register with the 'Postman' so that 'Letters' sent to them can be delivered.

Intra-agent communication

Roles within an agent might need to communicate between them to exchange information and/or to delegate the computation of tasks. For example, within the AOC agent, the Airline Flight Planner role might require the Airline Passenger Handler to reallocate passengers if it decides to cancel a flight.

In some cases, the protocols of the role that need to be accessed are also services from the agent. That is, they can be accessed by other agents with an external message. In that case, two options are possible: either the initiator role can directly communicate with the targeted role with a direct call, or a self-message can be sent to the agent (as explained in Section 4). As it is assumed that the roles of an agent have access to all the information of such an agent, the former approach is preferred for computational performance, as the number of messages and interactions is reduced.

5 Evaluation of ATM Solutions

Mercury can evaluate ATM Solutions by modifying the behaviour of the system's elements. 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 using 'Modules',
2. following a microservice approach, connecting external systems to replace roles and/or agents from Mercury.

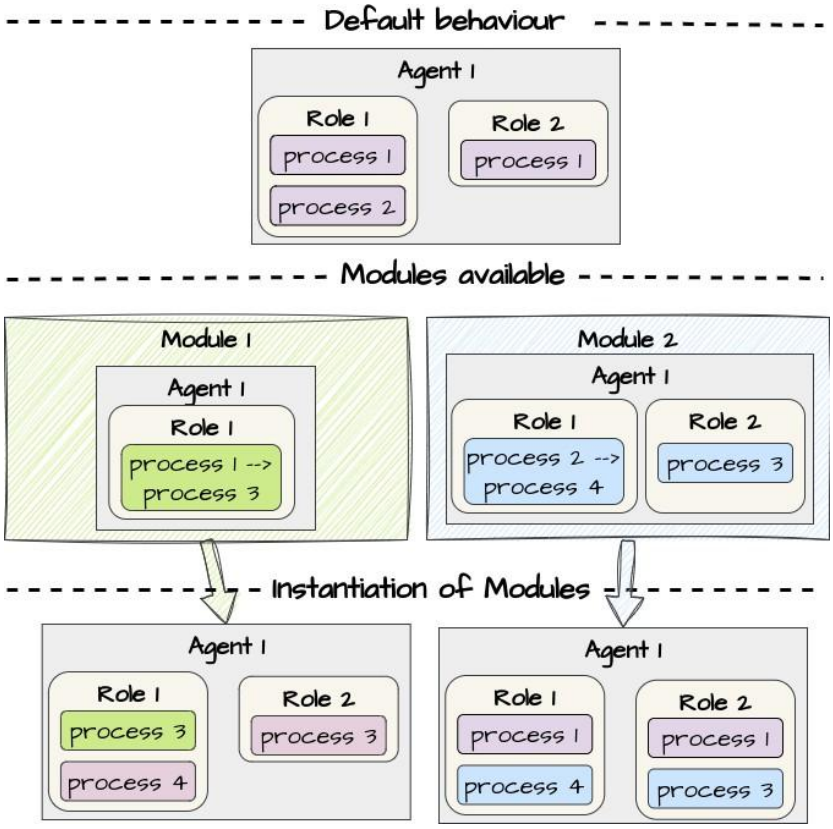

Figure 3. Example of Modules modifying the roles of an agent. From the default behaviour described by two roles, two modules are available: Module 1, which replaces process 1 of Role 1 for process 3; and Module 2, which replaces process 2 of Role 1 by process 4 and adds a new process (process 3) to Role 2. The third level represents the obtained agents if Module 1 or Module 2 is instantiated.

5.1 Modules

A module is a piece of code written outside the core source code that can be loaded at runtime and modifies the model's behaviour. Indeed, 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/or added to the agents.

Figure 3 presents an example of the principles of this approach, by describing agent default behaviour, available modules, and finishing with the examples of Agent 1 if the modules are instantiated. A *default behaviour* of a given agent (Agent 1) has two roles (Roles 1 and 2) with associated processes (process 1 and 2 in Role 1 and process 1 in Role 2). Two *modules are available* for modification of the agent's behaviour: Module 1, which provides a different implementation for process 1 of Role 1 (as a Process 3), and Module 2, which provides a new implementation of process 2 of Role 1 (as a Process 4) and an additional process to be added to Role 2 (process 3). Two possible *instantiations* are depicted in the last row of the figure, as described in the previous sentence.

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 landing slots available to the flight. With that information, the E-AMAN then assigns the slot for the flight, aiming to minimise its expected cost. Instead, the first slot available could 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 to the agents as required. 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 (modifying more than one agent at once). The code within the modules must be compatible with the remaining software, and developing these modules might require detailed knowledge of Mercury's inner workings.

5.2 Microservices approach

Another approach that can be followed to evaluate external components with Mercury and change 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 to 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 (Dragoni et al., 2017).

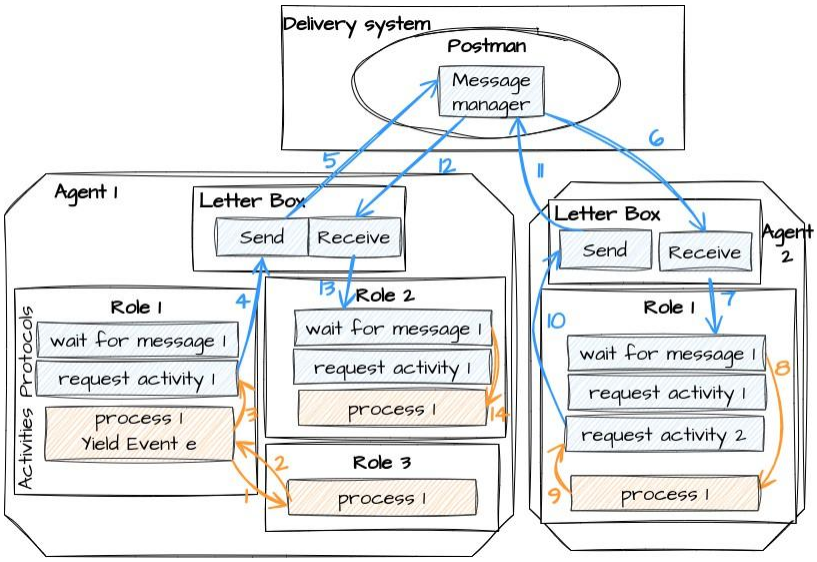

Figure 4. Example of communication between Agents and Roles in Mercury.

This approach allows for different technologies of 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.

A relatively complex example of interactions within Mercury elements, consisting of two agents and four roles, is presented in Figure 4. This example illustrates some of the characteristics of the communication channels within Mercury: intra- and inter-agents, as well as 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 information from Process 1 of Role 3; as both roles 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 computation, requests an action from Agent 1 (9 to 12). In this case, the

message from Agent 2 is treated by Role 2 within Agent 1 (13), which triggers the activities of Process 1 in that role (14). This process could, for instance, then wait until another event is satisfied.

Figure 5 below, presents the same interactions, where several elements are 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 Role 3 functionalities of Agent 1. **First**, the inter-agent communication mechanism can be used to create a message and send it to the Delivery system, which will redirect it outside Mercury. Second, Role 3 can manage the external communications, maintaining the direct interaction between Role 1 and Role 3. The **second case** is illustrated in Figure 5. 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 the externalisation of some internal processes within a role rather than the entire 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, e.g., with internal ad-hoc events. 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 (Gurtner & Bolic, 2023a) 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.

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

Mercury has this concept of external communications already implemented and validated, replacing some functionalities (selection of flight plans and prioritisation of flights in UDPP situations) with decisions coming from external actors interacting with the simulator through a dedicated interface enabling human-in-the-loop simulations (Gurtner & Bolic, 2023a).

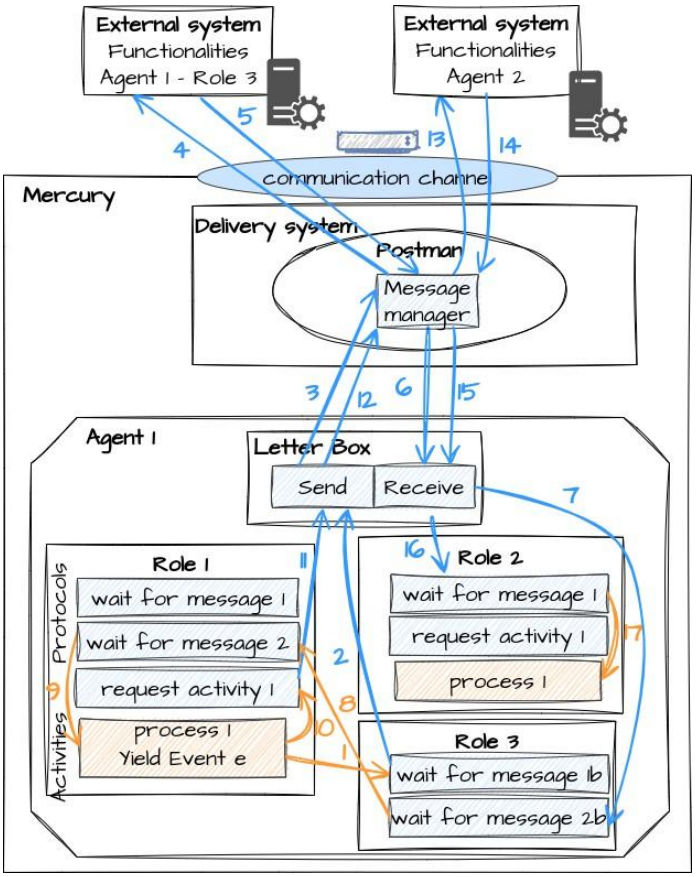

Figure 5. The example of communication, as in Figure 4, with external elements replacing systems from Mercury.

6 Data and configuration

Configuration files are defined using TOML format. These files configure some of the characteristics of Mercury and the simulations and define the modules' characteristics. An example can be seen in Figure 6.

```
[write_profile]
type = "file" # 'file' or 'mysql. Put "None" if you don't want a raw output (note: independent of aggregator)
fmt = "csv" # csv or pickle
connection = "local" # None if creds are included in read. 'local' for local saving of files
mode = "replace" # Can be update or replace
path = "../results"  # path folder where to save results.
                     # relative path are relative to base_path from profile file
                     # Otherwise you can put absolute paths, it will override base_path
prefix = "model_scenario_cs_it" # Destination folder for output files
```

Figure 6. Example of TOML file structure used for configuration files.

6.1 Input data

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

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.

The dataset is structured in the following ten categories:

- schedules: containing the flight schedules.
- passengers: containing the passengers' itineraries for the flight schedules.
- airports: with airport static and operational information, such as their location, but also curfew information, taxi times, and capacities.
- airlines: containing static information about the airlines operating the scheduled flights, like type (low cost, legacy) and the alliance they are part of.
- aircraft performance: with information on the flight performance data.
- flight plan: which contains a pool of pre-computed flight plans for each origin-destination and aircraft type available in the schedules, along with other operational data such as distribution of average en-route winds.
- delay: containing information on primary delays not due to ATFM.
- eaman: including information on which airports operate this extended arrival system and the scope of their planning and tactical horizons.
- cost: with information related to the modelling of the cost of delay (curfew, passengers' compensation, duty of care, etc.).
- network: containing all the information regarding ATFM delay (probabilities, distributions, and details of historical ATFM regulations at airports).

For some of these data, note that not only is a given value necessarily provided, but different alternatives could be available. For example, the probability of being regulated by ATFM delay could be provided at different levels: 'High', 'Medium', and 'Low'; these can be computed as representatives of historical days. These probability distributions are paramount for calibrating the model (Gurtner et al., 2021).

Case Study

A Scenario can contain 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 selected flights, i.e., a subset of passenger itineraries, airports, etc.

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

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

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 pre-defined probability distributions, which need to be calibrated with historical datasets as described in Annex 0. Therefore, several simulator runs are needed to obtain statistically significant results. The configurations of the execution of Mercury, along with information on computational settings, like the number of parallel executions, need to be defined in the experiment and the simulation configuration files.

6.2 Output data

The output data maintains traceability with the inputs 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 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 a 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) (Delgado et al., 2020), network complexity parameters (Zaoli et al., 2019), or analysing the emergent agents' behaviour (Gurtner et al., 2021).

7 Usage

7.1 Computational performance

The computational requirements continue to be contained, with a simulation of a whole day of operations in Europe requiring around 9GB of RAM and 30 minutes of computation (on a CPU@3.2GHz, using one core). Moreover, as presented in (Gurtner et al., 2021), the resources per individual flight decrease as the number of flights increases, showing good scalability. Note that in general, because the simulator is stochastic, several independent runs are performed on the same input, in order to compute proper statistics, see Annex 0. Mercury includes the possibility to make these runs in parallel on several cores out-of-the box.

7.2 User interface

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

A web-based graphical interface that connects to these managers has been developed, providing basic functionalities to explore the input and output data and create the case studies, experiments, and configuration of Mercury.

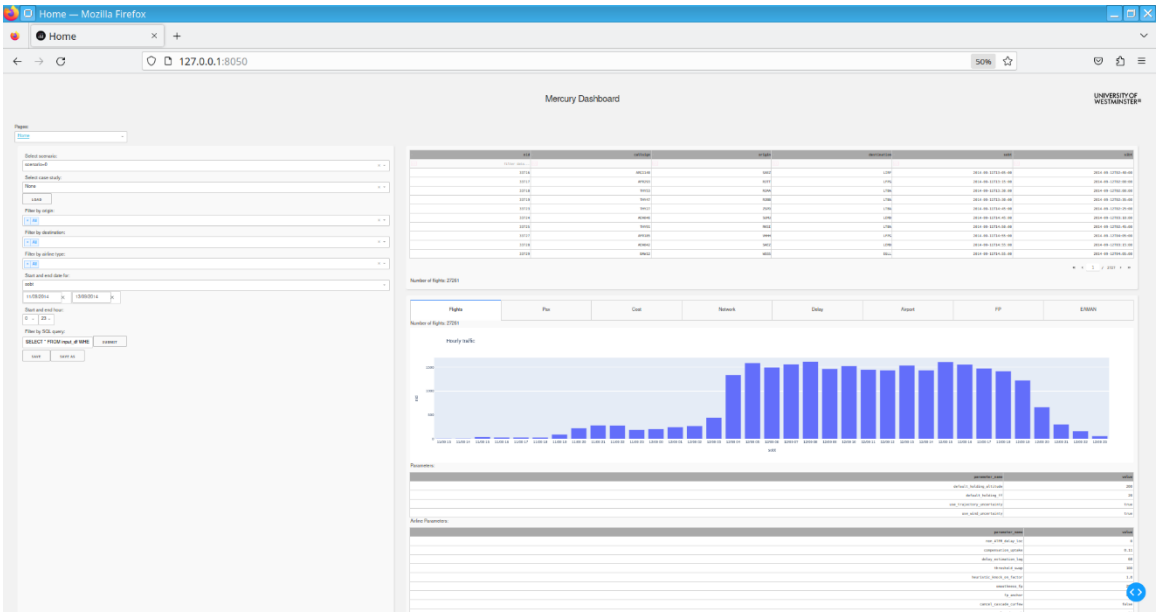

Figure 7. Web-based user interface.

Figure 7 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. The Case Studies can be loaded, if available, within a given Scenario.

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 8 shows the visualisation of the flight plan pool filtered by a given origin and destination. In some cases, as in Figure 8, 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).

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 9. Note how not only average results per KPI are produced, but also percentiles and distribution of these can be estimated.

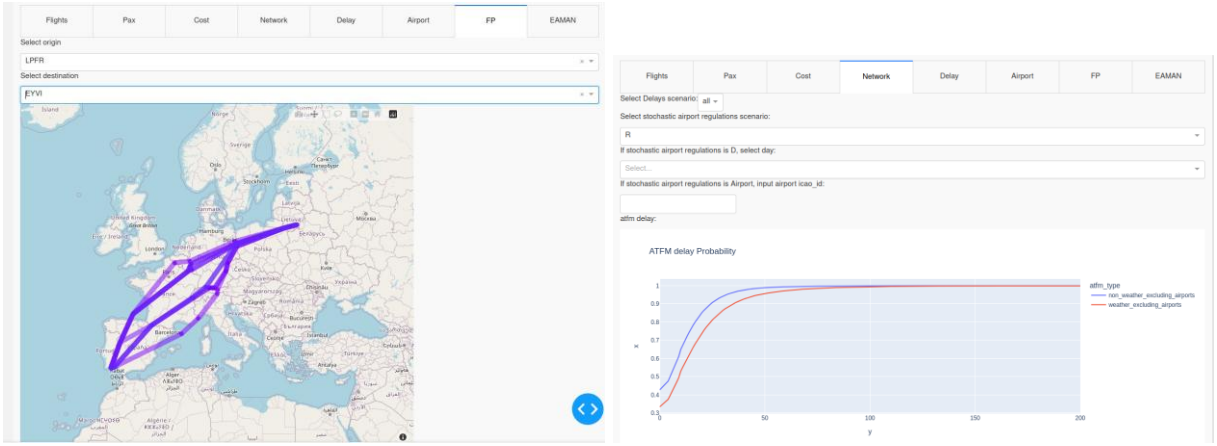

Figure 8. Left: flight plan alternatives exploration interface. Right: network-ATFM data visualisation and selection.

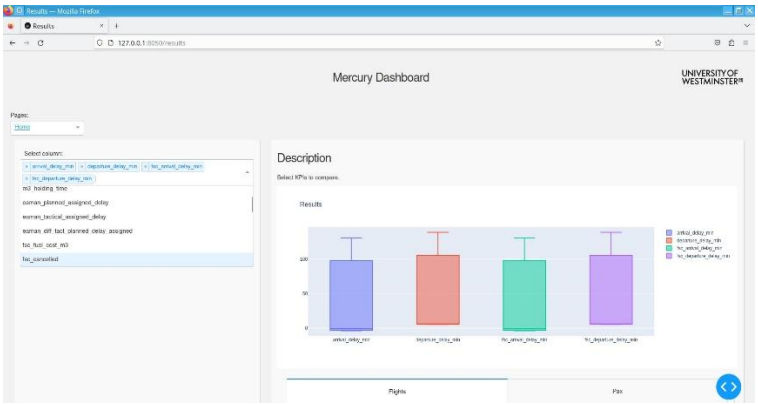

Figure 9. Results analysis and visualization.

8 Current and future development

Current developments aim to increase Mercury's flexibility of use, support the integration with other systems, and extend the model to consider multimodal journeys.

8.1 Input/Output interface improvements

The interface for manipulating input and output datasets (presented in Section 7.2) is being 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, with a particular effort devoted to improving the automatic analysis and presentation of the simulations' output. This is relevant considering the large number of low-level indicators that can be obtained from a simulation and the stochastic nature of the model (see Annex 0), which requires several runs (following a Monte Carlo approach) to obtain statistically representative results.

In parallel, the datasets (input/output) are being fully documented, providing the required metadata to develop new scenarios.

8.2 Integration with other systems

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 through interaction with an external interface (Gurtner & Bolic, 2023a). These efforts will continue to develop a full messaging API that will enable 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 (Delgado et al., 2023).

Future work will aim to standardise the usage of modules and external systems integration, facilitating the wider usage of Mercury.

Mercury's external communication system (described in Section 4.2) is being standardised by using message brokers, such as RabbitMQ (RabbitMQ, 2023), and standardisation of messages schema, e.g., using Apache Avro (Apache, 2023). This should facilitate the integration of external functionalities, such as disruption management solvers for traffic. The MultiModX project provides a range of use cases and challenges that will contribute to these developments: from simple request-reply messages, such as from an external system to obtain the transfer times between train stations and airports (Weiszer et al., 2024), to more advanced tools, such as solvers to manage passengers and flights under system disruptions which might require various exchanges between Mercury and the external tool to obtain the information and produce the optimisation, as in (Scozzaro et al., 2023), or even potentially connecting Mercury to external simulators, such as a rail network simulator, with the synchronisation challenges that would arise.

8.3 Beyond gate-to-gate mobility

As part of MultiModX (CORDIS, 2026), Mercury is being extended to model the passenger journey, including rail multimodal segments. The approach to model multimodal itineraries is to divide the passenger journey into segments which represent the different transport phases: platform-to-platform (for rail segment), platform-to-kerb (for ground mobility transfer between train station and airport), kerb-to-gate (for airport terminal passengers processes such as check-in and security control), gate-to-gate (considered in the current implementation of Mercury), etc. (Delgado et al., 2022). This approach enables the simulation of gate-to-gate (including connections) times for air-only passengers, platform-to-gate times for rail-air itineraries, and gate-to-platform times for air-rail passengers. These travelling times could then be extended to estimate door-to-door mobility as post-processing, e.g., as done in (Cook et al., 2017; Delgado et al., 2022; Gurtner et al., 2018).

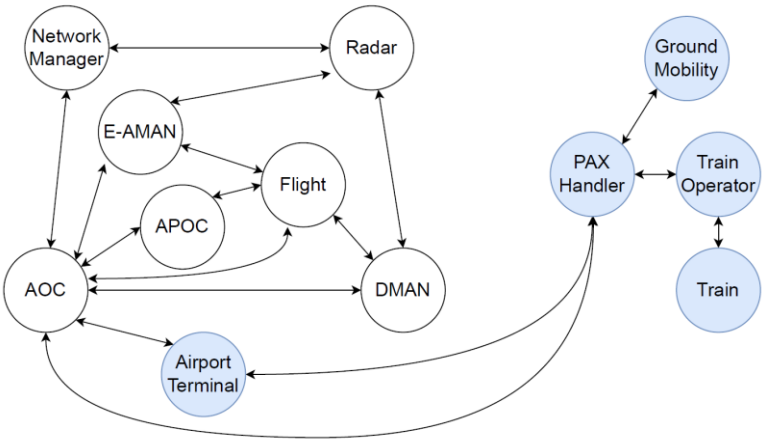

Figure 10. Acquaintances between agents with extension to multimodality.

As shown in Figure 10 the model has been modified and extended to consider these processes with their associated decision points.

First, the Ground Airport agent has been divided between an Airport Operating Centre (APOC), which maintains the roles in charge of the flight-related processes, like turnaround time estimation, and the Airport Terminal agent, which encompasses the activities associated with moving passengers through the terminal (connecting times and kerb-to-gate and gate-to-kerb times estimation and realisation). Then, new dedicated agents were added:

- Passenger Handler: This agent represents the decision processes of a passenger when transferring across modes. For example, which mode of transport to use to move between the train station and the airport (and vice-versa), considering accrued delays and schedules and ground mobility times, if rebooking the itinerary before arriving at the airport or train station, as it estimates the connection will be missed, etc. From an architectural point of view, there is a single passenger handler agent in the simulation that manages passenger groups.
- Train: The train agent models the processes related to the operation of a train, mainly arrival and departure from stations.
- Train Operator: This agent keeps track of the arrivals and departures of trains and any train delays. It is responsible for the passengers' boarding and deboarding, and, in case of missed connections, it can rebook passengers on the next direct train.
- Ground Mobility: The connection between the train station and the airport is provided by this agent. Ground mobility can provide expected (and actual) transfer times as a function of transport mode (e.g., taxi, bus metro).

New events are incorporated into Mercury to simulate the multimodal journeys, notably: train arrival and train departure. Note that this extension of Mercury does not impede the use of Mercury just for the gate-to-gate simulation, i.e., without any multimodal consideration.

8.4 Simulation performance aspects

The simulation of the gate-to-gate processes of all flights of a nominal day of operations in Europe requires around 30 minutes of computation with 9GB of RAM (on a CPU@3.2GHz, using one core). This level of performance is sufficient for a research platform such as Mercury, as used in current and previous research projects. However, gains could be obtained by optimising parts of the code. This might be required if more complex mechanisms (e.g., dynamic cost indexing reoptimisation, as in (Delgado et al., 2024)) or ATM aspects (e.g., including airspace management) are to be evaluated with Mercury, as the number of events and processes could increase significantly.

These performance aspects are continuously under review, and computationally demanding functionalities (e.g., estimation of cost of delay in some situations) could be reimplemented relying on higher-performance libraries (e.g., with ad-hoc implementation in C/C++).

The SimPy library provides the required framework for discrete event-driven simulation, as the one performed in Mercury; other approaches, such as developing an ad-hoc simulation engine relying on high-performance asynchronous libraries such as uvloop (MagicStack, 2023), could be explored. However, a trade-off exists between the effort required for these changes and developments and the potential computational benefits that could be obtained.

9 Conclusions

Previous research results have proven that Mercury can be used to integrate new behaviours and mechanisms for their evaluation, considering network effects, complex interactions, and 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.

Opening Mercury will allow its evolution 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.

Funding

The development of this version of Mercury has been funded by various projects:

- H2020, Novel tools to evaluate ATM systems coupling under future deployment scenarios (Domino) project, grant number 783206
- H2020, Behavioural Economics for ATM Concepts (BEACON) project, grant number 893100
- H2020, Next-generation Open-Source Tools for ATM performance Modelling and Optimisation (NOSTROMO) project, grant number 892517
- Horizon Europe, Integrated Passenger-Centric Planning of Multimodal Transport Networks (MultiModX), grant number 101114815

Data and Software Access Statement

The code source of Mercury can be found here: https://github.com/UoW-ATM/Mercury

A synthetic dataset usable by Mercury of the shelf can be found here: https://zenodo.org/records/11384379

Author and Contributor Statement

Conceptualization: Luis Delgado, Gérald Gurtner, Michal Weiszer; Methodology: Luis Delgado, Gérald Gurtner, Michal Weiszer; Data Curation: Luis Delgado, Gérald Gurtner, Michal Weiszer, Tatjana Bolić; Visualization: Luis Delgado, Michal Weiszer, Gérald Gurtner; Writing – Original Draft: Luis Delgado, Michal Weiszer, Gérald Gurtner, Tatjana Bolić, Andrew Cook; Writing – Review & Editing: Luis Delgado, Michal Weiszer, Gérald Gurtner, Tatjana Bolić, Andrew Cook

Use of AI

During the preparation of this work, the authors did not use AI.

Acknowledgements

Mercury has been supported by various additional projects:

- SESAR Joint Undertaking, Passenger-Oriented Enhanced Metrics (POEM) project, project number E.02.06
- SESAR Joint Undertaking, ComplexityCosts project, project number E.02.35
- H2020, Data driven approach for a Seamless Efficient Travelling in 2050 (Dataset2050), grant number 640353
- H2020, Coordination and support Action for Mobility in Europe: Research and Assessment (CAMERA) project, grant number 769606
- H2020, Market forces trade-offs impacting European ATM performance (Vista) project, grant number 699390
- H2020, Modelling and assessing the role of air transport in an integrated, intermodal transport system (Modus), grant number 891166

Conflict Of Interest (COI)

There is no conflict of interest.

References

Apache. (2023). *Apache Avro* (Version 1.11.3) [Computer Software]. https://avro.apache.org/

Barnhart, C., Fearing, D., & Vaze, V. (2014). Modeling passenger travel and delays in the national air transportation system. *Operations Research*, *62*(3), 580–601. https://doi.org/10.1287/opre.2014.1268

Bratu, S., & Barnhart, C. (2005). An analysis of passenger delays using flight operations and passenger booking data. *Air Traffic Control Quarterly*, *13*(1), 1–27. https://doi.org/10.2514/atcq.13.1.1

Cook, A., Delgado, L., Tanner, G., & Cristóbal, S. (2016). Measuring the cost of resilience. *Journal of Air Transport Management*, *56*, 38-47. https://doi.org/10.1016/j.jairtraman.2016.02.007

Cook, A., & Tanner, G. (2015). *European airline delay cost reference values: Updated and extended values (Version 4.1)*. University of Westminster. https://www.eurocontrol.int/sites/default/files/publication/files/european-airline-delay-cost-reference-values-final-report-4-1.pdf

Cook, A., Tanner, G., Cristóbal, S., & Zanin, M. (2012). Passenger-oriented enhanced metrics. *Second SESAR Innovation Days*.

Cook, A. J., Tanner, G., Gurtner, G., Ureta, H., Cristobal, S., Belkoura, S., Gómez, I., Paul, A., Kluge, U., & Hullah, P. (2017). *DATASET2050 D5.2 - Assessment execution*. https://westminsterresearch.westminster.ac.uk/item/q4v11/dataset2050-d5-2-assessment-execution

CORDIS. (2022a). *Coordination and support Action for Mobility in Europe: Research and assessment (CAMERA) project*. Publications Office of the European Union. https://doi.org/10.3030/769606

CORDIS. (2022b). *Data driven approach for a seamless efficient European travelling in 2050 (DATASET2050) project*. Publications Office of the European Union. https://doi.org/10.3030/640353

CORDIS. (2022c). *Market forces trade-offs impacting European ATM performance (Vista) project*. Publications Office of the European Union. https://doi.org/10.3030/699390

CORDIS. (2023a). *Behavioural economics for ATM concepts (BEACON) project*. Publications Office of the European Union. https://doi.org/10.3030/893100

CORDIS. (2023b). *Next-generation open-source tools for ATM performance modelling and optimisation (NOSTROMO) project*. Publications Office of the European Union. https://doi.org/10.3030/892517

CORDIS. (2025). *Novel tools to evaluate ATM systems coupling under future deployment scenarios (Domino) project*. Publications Office of the European Union. https://doi.org/10.3030/783206

CORDIS. (2026). *Integrated passenger-centric planning of multimodal transport networks (MultiModX) project*. Publications Office of the European Union. https://doi.org/10.3030/101114815

Delgado, L., Blanch, A., Cristóbal, S., Martín, J. (2016). *CASSIOPEIA II D3.2 - Final technical report*. https://westminsterresearch.westminster.ac.uk/item/q4v68/cassiopeia-ii-d3-2-final-technical-report

Delgado, L., Bolic, T., Cook, A. J., Zareian, E., Gregori, E., & Paul, A. (2023). Modelling passengers in air-rail multimodality. *Proceedings of the 11th EUROSIM Congress*.

Delgado, L., Cook, A., Zareian, E., Bolic, T., Gregori, E., & Paul, A. (2022). Challenges of multimodal door-to-door mobility modelling. *12th SESAR Innovation Days*.

Delgado, L., Cristobal, S., Cook, A. J., & Tanner, G. (2016). *ComplexityCosts D4.5 - Final technical report. https://westminsterresearch.westminster.ac.uk/item/q4v26/complexitycosts-d4-5-final-technical-report*

Delgado, L., de la Torre, D., Kuljanin, J., & Prats, X. (2024). Considering expected TMA holding into in-flight trajectory optimization. *Transactions of the Japan Society for Aeronautical and Space Sciences*, 67(3), 109-118. https://doi.org/10.2322/tjsass.67.109

Delgado, L., Gurtner, G., Cook, A., Martín, J., & Cristóbal, S. (2020). A multi-layer model for long-term KPI alignment forecasts for the air transportation system. *Journal of Air Transport Management*, 89, Article 101905. https://doi.org/10.1016/j.jairtraman.2020.101905

Delgado, L., Gurtner, G., Cook, A. J., Pilon, N., Valput, D., Cristobal, S., & Tanner, G. (2018). *Domino D4.1 - Initial model design.* https://westminsterresearch.westminster.ac.uk/item/q9q47/domino-d4-1-initial-model-design

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, today, and tomorrow. In M. Mazzara & B. Meyer (Eds.), *Present and Ulterior Software Engineering* (pp. 195–216). Springer. https://doi.org/10.1007/978-3-319-67425-4_12

EUROCONTROL. (n.d.-a). *DDR: Demand data repository*. Retrieved January, 2025, from https://www.eurocontrol.int/ddr

EUROCONTROL. (n.d.-b). *R-NEST: Research network strategic monitoring tool*. Retrieved January, 2025, from https://www.eurocontrol.int/solution/rnest

European Commission. (2004). *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. 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* [Document No. 32004R0261]. http://data.europa.eu/eli/reg/2004/261/oj

European Commission: Directorate-General for Research and Innovation & Directorate-General for Mobility and Transport. (2011). *Flightpath 2050: Europe's vision for aviation: Maintaining global leadership and serving society's needs*. Publications Office of the European Union. https://www.doi.org/10.2777/50266

Gurtner, G., & Bolic, T. (2023a). *Deliverable D5.2: Final tactical model and results*. https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5f7897f46&appId=PPGMS

Gurtner, G., & Bolić, T. (2023b). Impact of cost approximation on the efficiency of collaborative regulation resolution mechanisms. *Journal of Air Transport Management*, *113*, Article 102471. https://doi.org/10.1016/j.jairtraman.2023.102471

Gurtner, G., Delgado, L., & Valput, D. (2021). An agent-based model for air transportation to capture network effects in assessing delay management mechanisms. *Transportation Research Part C: Emerging Technologies*, *133*, Article 103358. https://doi.org/10.1016/j.trc.2021.103358

Hoekstra, J. M., & Ellerbroek, J. (2016). BlueSky ATC simulator project: An open data and open source approach. In *7th International Conference on Research in Air Transportation*.

Kluge, U., Paul, A., Ureta, H., & Ploetner, K. O. (2018). Profiling future air transport passengers in Europe. *Proceedings of 7th Transport Research Arena TRA 2018*.

MagicStack. (2023). *uvloop* (Version 0.18.0). https://uvloop.readthedocs.io/

Mazzarisi, P., Zaoli, S., Lillo, F., Delgado, L., & Gurtner, G. (2020). New centrality and causality metrics assessing air traffic network interactions. *Journal of Air Transport Management*, *85*, Article 101801. https://doi.org/10.1016/j.jairtraman.2020.101801

Nuic, A., Poles, D., & Mouillet, V. (2010). BADA: An advanced aircraft performance model for present and future ATM systems. *International Journal of Adaptive Control and Signal Processing*, *24*(10), 850-866. https://doi.org/10.1002/acs.1176

RabbitMQ. (2023). RabbitMQ -- One broker to queue them all (Version 3.11.22). https://www.rabbitmq.com

Riis, C., Antunes, F., Bolić, T., Gurtner, G., Cook, A., Azevedo, C. L., & Pereira, F. C. (2024). Explainable active learning metamodeling for simulations: Method and experiments for ATM performance assessment. *Transportation Research Part C: Emerging Technologies*, *166*, Article 104788. https://doi.org/10.1016/j.trc.2024.104788

Scozzaro, G., Mancel, C., Delahaye, D., & Feron, E. (2023). An ILP approach for tactical flight rescheduling during airport access mode disruptions. *International Transactions in Operational Research*, *31*(3), 1426-1457. https://doi.org/10.1111/itor.13396

SimPy. (2023). SimPy -- Discrete-even simulation for Python (Version 4.1.1). https://simpy.readthedocs.io/en/latest/index.html

Sun, J., Hoekstra, J. M., & Ellerbroek, J. (2020). OpenAP: An open-source aircraft performance model for air transportation studies and simulations. *Aerospace*, *7*(8), Article 104. https://doi.org/10.3390/aerospace7080104

Weiszer, M., Delgado, L., & Gurtner, G. (2024). Multimodal air-rail simulation model for evaluation of tactical disruptions. *27th Air Transport Research Society (ATRS) World Conference*.

Wooldridge, M., Jennings, N. R., & Kinny, D. (2000). The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, *3*(3), 285–312. https://doi.org/10.1023/A:1010071910869

Zaoli, S., Mazzarisi, P., & Lillo, F. (2019). Trip centrality: Walking on a temporal multiplex with non-instantaneous link travel time. *Scientific Reports*, *9*, Article 10570. https://doi.org/10.1038/s41598-019-47115-6

Annex A – Probability distributions in Mercury

Mercury relies on the modelling of several processes with some degree of uncertainty to capture a day of operations realistically. These processes need to be calibrated using historical datasets.

Table 3 presents the list of stochastic processes considered in Mercury. The Table also provides which distributions are used to model these, and the datasets used in previous projects for their estimation (Gurtner et al., 2021).

Note that when creating case studies and experiments, the end user could modify some of these distributions to evaluate their impact on the performance of the system. For example, estimating distributions of ATFM delays based only on historical days with low capacity, so that the performance of a given mechanism can be evaluated under those conditions. In some other cases, if required, ad-hoc values could be evaluated (e.g., manually defining ATFM regulations at airports, or larger than usual taxi or turnaround times).

Table 3. Probabilistic parameters to calibrate in Mercury (Gurtner et al., 2021).

Process	Distribution	Based on
Taxi-in/out	LogNormal distribution (minutes)	IATA Summer Season 2010 from CODA
Climb uncertainty	Normal distribution (minutes)	Analysis DDR2 difference between planned and executed trajectories (M2, M3) from DCI-4HD2D Project (Delgado et al., 2016)
Cruise	Normal distribution (Nautical miles)	Analysis DDR2 difference between planned and executed trajectories (M2, M3) from DCI-4HD2D Project (Delgado et al., 2016)
Wind	Empirical probability distribution for planned wind during the cruise. Used percentile of wind between regions. No noise added on execution.	For each ANSP to ANSP origin and destination airport consider the difference between requested speed and observed average ground speed for cruise segments from DDR2 analysis (AIRAC1409).
Turnaround time	Exponential distribution based on minimum turnaround time as a function of airport size, aircraft wake turbulence category, and type of airline.	Analysis of turnaround times performed in POEM project and used in ComplexityCosts project (Delgado et al., 2016)
Airport ATFM delay	Airport regulations are sampled from a historical day. The day is selected based on their percentile ranked by the number of regulations at the airport on that day.	Based on analysis of DDR2 (AIRAC1313-1413 excluding days with industrial actions (EUROCONTROL, n.d.-a).
Airspace ATFM delay	Two empirical probability distribution functions are used to model the ATFM delay due to airspace regulations: one for regulations due to weather, and other regulations for other reasons	Based on analysis of DDR2 (AIRAC1313-1413 excluding days with industrial actions) (EUROCONTROL, n.d.-a)
Non-ATFM delay	Exponential distribution	Calibrated
Connecting times	LogNormal distribution based on minimum connecting times per airport and type of connection (national/international)	Based on analysis of minimum connecting times at ECAC airports originally performed in POEM project (Cook et al., 2012)
Variation of cruise length due to dynamic cost indexing	Normal distribution (Nautical Miles)	Analysis of Performance using Airbus PEP (Delgado et al., 2016)

Annex B – Software characteristics

This Annex describes some details on the software characteristics of Mercury: programming language and libraries used, data manipulation and system modification, and software architecture.

Programming language and libraries

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

The model is developed from scratch but relies on third-party libraries like Simpy, for the discrete-event simulation framework, pandas6, for data structure and manipulation, and NumPy, for fast numerical computations on arrays and matrices, all open-source.

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

- Hotspot[7]: developed as part of BEACON project (CORDIS, 2023a) to manage hotspots and air traffic flow management (ATFM) regulations for UDPP concepts. Open-source under GPL v3.
- uow-tool-belt[8]: a library providing basic tools and functionalities, like read/write functionalities. Open-source under GPL v3.
- EUROCONTROL's BADA (Nuic et al., 2010): this aircraft performance model can be used by Mercury. To use BADA3, users need to ask for a licence, which is normally granted for free by EUROCONTROL, and can input the coefficients directly into the model. To use BADA4, users have to ask for a licence too and build an interface equivalent to the one included for BADA3 to make use of the model in Mercury.
- OpenAP (Sun et al., 2020): an open-source alternative to EUROCONTROL's BADA model for aircraft performance. OpenAP is included in the repository and is the default option for the aircraft performance in Mercury.

Data manipulation and system modification

To facilitate the input and output manipulation, Mercury uses the open-source Apache Parquet format, but also allows the use of other formats as a source (CSV, MySQL database). Data is compressed efficiently in tables that are easy to analyse and manipulate while maintaining properties such as their data types. See Section 6 for more information about the data used and generated by Mercury.

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

Software architecture

Mercury is organised in three packages:

Agents

Agents is the main package containing the implementation of different agents in Mercury. The agents are developed following an object-oriented approach. Each agent type is a Class containing its memory (attributes) and Roles. The Roles are independent Classes contained within the Agents.

---

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

All agent types inherit from a generic Agent class, which provides the shared functionalities of initialisation, mailbox, and functionalities required to modify their behaviour through the application of Modules. Two sub-packages are located inside the agents' package:

- Modules: This package stores different modules that can be loaded into Mercury. A Module is composed of three files:
    - the Python code implementing the functionalities that need to be added and/or replaced in the different Roles,
    - a configuration file indicating which functions need to be added/replaced for which roles, and
    - an optional configuration file with any additional parameters needed for the new functionalities implemented in the module.
- Commodities: Contains different objects used and manipulated by the agents, such as the definition of aircraft, alliance, slots, etc. Each one of these concepts will be represented by one or several classes.

libs

The libs package contains functionalities required by Mercury, such as the implementation of the Delivery system, World builder (to create the agents at the instantiation of a simulation), Simulation manager (to manage the execution of Mercury), Case study loader, etc. Functionalities to manage the input and output of Mercury are also provided here (Input and Output managers). Finally, external libraries are also included here.

config

The config package contains the configuration files of Mercury and the simulations.