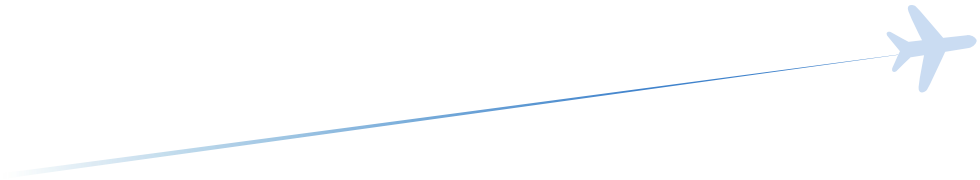

D4.1 Initial model design

Deliverable 4.1

Domino

Grant:	**783206**
Call:	**H2020-SESAR-2016-2**
Topic:	**SESAR-ER3-06-2016 ATM Operations, Architecture, Performance and Validation**
Consortium coordinator:	**University of Westminster**
Edition date:	**08 November 2018**
Edition:	**01.00.00**

Founding Members
EUROPEAN UNION EUROCONTROL
SESAR JOINT UNDERTAKING

Authoring & Approval

Authors of the document

Name/Beneficiary	Position/Title	Date
Luis Delgado / University of Westminster	Project member	06 November 2018
Gérald Gurtner / University of Westminster	Project member	06 November 2018
Andrew Cook / University of Westminster	Project coordinator	06 November 2018
Damir Valput / Innaxis	Project member	06 November 2018
Samuel Cristóbal / Innaxis	Project member	06 November 2018

Reviewers internal to the project

Name/Beneficiary	Position/Title	Date
Graham Tanner / University of Westminster	Project member	08 November 2018

Approved for submission to the SJU By — Representatives of beneficiaries involved in the project

Name/Beneficiary	Position/Title	Date
Andrew Cook / University of Westminster	Project coordinator	08 November 2018

Rejected By - Representatives of beneficiaries involved in the project

Name/Beneficiary	Position/Title	Date
N/A		

Document History

Edition	Date	Status	Author	Justification
01.00.00	08 November 2018	Release	Domino Consortium	New document for review by the SJU

Domino

NOVEL TOOLS TO EVALUATE ATM SYSTEMS COUPLING UNDER FUTURE DEPLOYMENT SCENARIOS

This deliverable is part of a project that has received funding from the SESAR Joint Undertaking under grant agreement No 783206 under European Union's Horizon 2020 research and innovation programme.

Abstract

This technical deliverable describes the model design used in Domino. Domino deploys an agent-based model that has been developed following the Gaia methodology. This deliverable contains the requirements, specification and design of the model. This includes the definition of the roles and interactions models as part of the analysis of the system, and the agent, services and acquaintance models as part of the design of the model.

Other design issues such as the simulation engine, communication channels and potential parallelisation are described. This deliverable also presents some implementation details such as the programming language and potential libraries that will be used.

Table of Contents

List of figures

List of tables

Executive summary

D4.1 presents the design of the model that will be used to simulate the different scenarios that Domino will analyse.

The modelling approach is presented (from p11), in which the rationale for an agent-based model is described. The specification of the model follows the design methodology known as Gaia, which allows one to start from the requirements of the model in order to specify the roles needed to meet these requirements. The agents are collections of roles. A detailed specification of the interactions between roles is also planned by the methodology.

Requirements are then laid out (from p16). Most are linked to the type of scenarios that Domino will simulate, and, in particular, the mechanisms presented in D3.2 (4D trajectory adjustments, flight prioritisation, flight arrival coordination) [2]. Others are linked to the type of output that Domino will need to analyse and its granularity. In particular, information on the cost of fuel, detailed passenger connections, delays, and airline networks are required.

The analysis of the system is then performed (from p18). 38 distinct roles are defined, with their associated activities (computations they need to perform to do a given task) and protocols (the way they interact with other roles). After that, the nature of the interactions between roles is defined (59 distinct interactions are described).

The model is then designed, in terms of agents (from p36). Roles are grouped into agents (within which all variables are private). Nine distinct types of agent are identified. The services are then listed, which are the activities that each agent will perform during the simulation. These services are directly derived from the underlying roles. They have associated triggers, inputs, and outputs. The acquaintances of the agents are summarised: they represent the links of interactions between agents, arising directly from the underlying interactions between roles.

Other design issues, independent of the model specification itself, are then discussed (from p47). One of these is the type of simulation, chosen from: step-based, time-driven, event-driven, or message-driven. The Domino model will be event-driven. The main events are described, and sequence diagrams of the cascade of processes arising from their triggers are presented. These events are mainly generated from the milestones of a flight: flight plan submission, push-back, take-off, etc. A brief discussion of communication channels (for communication between agents) is presented, which discusses a few tests and the rationale for the chosen solution: a custom, dedicated server, which would allow further developments of the model in the future.

Some implementation details are then explained (from p60), concerning the type of language and the development flow. The document closes with the next steps for Domino.

Note that this document represents the specification and design of the model that will be implemented in Domino, but that divergences with the final implementation might exist as new issues might arise during the development and execution of the model.

1 Introduction

The model developed in Domino is an agent-based model (ABM) supported with an event-driven simulator. In this deliverable the specification and design of the model are presented. The deliverable contains also indications on some of the implementation aspects (such as the programming language, database description) that will be considered during the model development phase. Note that changes might be needed during the implementation of the model but this deliverable presents the current design that will be used to build the model.

1.1 Modelling approach

The objective of Domino is to develop a set of tools to assess the impact of operational changes in the ATM system. The tools are meant to be tested on case studies, for the scenarios which have been defined in D3.2 [2]. These scenarios have been designed with the double objective of demonstrating the capabilities of the tools developed in Domino while drawing some knowledge about the mechanisms defined in D3.1 [1]. These mechanisms are:

- 4D Trajectory Adjustments: collects several pro-active delay management behaviours like dynamic cost indexing, rerouting, etc.
- Flight Prioritisation: delay management techniques based on slot swapping (like User Driven Prioritisation Process, UDPP).
- Flight arrival coordination: different sets of rules used by the arrival manager to optimise the arrival queue.

The tools developed by Domino should be designed in such a way that they can be reused in the future. Thus, Domino's platform can be used as the basis for a future platform, which could resemble get close to an implementation of EATMA [3]. The main capability that Domino wants to demonstrate is the parallel but interdependent processing of a massive number of interactions between different entities leading to various system-wide outcomes. As explained in previous deliverables, Domino starts from the acknowledgement that macroscopic effects are grounded in a massive number of interactions between entities and processes, some of which trigger unforeseen effects. The necessity to take into account heterogeneity, uncertainty, dynamical effects, sub-optimal strategies and imperfect information leads us to build holistic models describing low-levels interactions, in order to extract knowledge from the system. This is crucial, when capturing the characteristics of the different elements in the system as envisioned by Domino.

This holistic modelling approach is particularly relevant when modelling the impact of changes in the system which divert significantly from known previous operations. Models designed to produce

knowledge based on past system states can be considered. These models will be mainly driven by data analysis and described by the term 'machine learning' techniques. However, the air transportation system features many known rules and processes, and other kinds of techniques can be used in conjunction with machine learning to use this prior knowledge.

1.1.1 Machine learning and agent-based modelling

Machine learning (ML) collects different techniques which, broadly speaking, infer rules (knowledge) from data. These data have usually been collected from a given system in the past. The inferred rules can then be used to predict the future behaviour of this system. As a consequence, ML can be used as models, i.e. as a set of mathematical relationships which, given an interpreted input, can give an interpretable output. In short, predict the future based on the past (in general).

There is no hard boundaries between pure ML and other techniques, but the main difference lies in the main they are built. Traditional highly data-driven models try to use only quantitative datasets to extract some correlation relationships between variables. On the other hand, causal modellers start by specifying causal relationships between variables using at first more qualitative data: their own knowledge of the system.

For instance, in a causal model, the modeller will explicitly state that when a controller asks to a pilot to descend, the pilot does it (some randomness could be added). Instead, a pure machine learning solution would be to track communication between controllers and pilots, and infer that in some situations a descent of the aircraft is consecutive to a message from the controller.

As a consequence, pure ML loses a lot of prior information that humans implicitly know, which need to be inferred from data. In fact, ML is even able to extract more meaningful relationships, since it is looking at statistically-optimal rules, which in general do not coincide with the human-defined ones. Of course, the strength of ML lies exactly there: that the human does not need to inject a priori knowledge, (potentially) biasing the model. Relationships then emerge spontaneously between variables.

Linked to this issue, ML usually infers correlations between variables, whereas causal models, as the name suggests, define mainly causal relationships. Another consequence is that ML models have, in some circumstances, poor extrapolation capabilities compared to causal models, because not so much qualitative data are used in them. This is particularly true in highly controlled systems like air transportation where lots of interactions are standard and thus known to the modeller. ML is thus extremely efficient for very specialised tasks where lots of quantitative data are available and little is known about the system.

This distinction should not hide the fact that ML techniques are usually used by causal models. For instance, the modeller can decide that at the airport level, traffic creates delay, and to use raw data to derive relationships which can be used quantitatively within the model. There is almost no limit to the degree to which one can combine different levels and type of models.

Agent-based modelling (also called massive multi-agents modelling when the number of agents is high) is a type of causal model where machine learning can be heavily used. It starts by defining generic roles and interactions between agents, but uses quantitative data to calibrate the way these agents behave. This is particularly well suited to the air transportation system where most of the

interactions between agents are well-known, but where more knowledge needs to be drawn from data about the details of these interactions. As a consequence, Domino has chosen from the start that the model will be agent-oriented. An additional feature of an agent-based model is that it is a 'white box', where anything happening can be in principle be traced back and understood by the modeller even if the behaviour of the agents could be driven by prior data analysis. This is contrast to some techniques of ML (neural networks, in particular), which act as 'black boxes' and thus are hard to apprehend for experts and non-experts alike.

1.2 Specification and design methodology

The specification and design methodology used for Domino for the agent-based model is Gaia [11]. Gaia is appropriate for the development of systems with large-scale applications with the characteristics shown below.

- Agents are coarse-grained computational systems, each making use of significant computational resources.
- It is assumed that the goal is to obtain a system that maximises some global quality measure(s), but which may be sub-optimal from the point of view of the system components. Gaia is not intended for systems that admit the possibility of true conflict (i.e. when there is no way to decide which situation is best).
- Agents are heterogeneous, in that different agents may be implemented using different programming languages, architectures, and techniques. There are no assumptions about the delivery platform.
- The organisational structure of the system is static, in that inter-agent relationships do not change at run-time.
- The abilities of agents and the services they provide are static, in that they do not change at run-time.
- The overall system contains a comparatively small number of different agent types (fewer than 100).

All these characteristics are met by Domino's model, however, we need to consider that even if the abilities and services provided by the agents do not change at run-time, they might change between different scenarios. Strictly speaking, we will need one model per scenario. However, these models will be very close to other, and we will focus on a single, main description, highlighting the different variants when necessary. Also, the system as a whole might not necessarily maximise an objective and some competition between agents might be present. These limitations will be solved in this deliverable by considering a flexible approach to the use of the modelling methodology in Domino.

Gaia deals with both the macro (societal) level and the micro (agent) level aspects of design. Gaia divides the different activities between two phases: analysis and design. The objective of the analysis is to develop an understanding of the system and its structure capturing the different roles and their interactions. The design phase transforms the abstract models derived from the analysis into models at the level of detail which allows their further implementation.

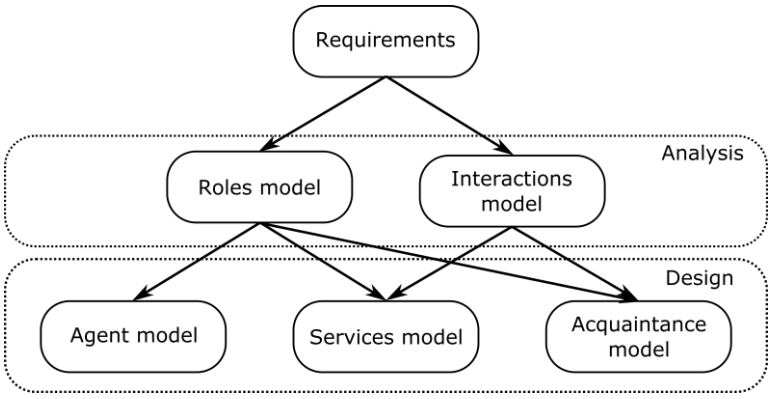

Figure 1: Relationships between Gaia's models (Based on [11])

As shown in Figure 1, from the requirements of the system a set of roles and their interactions are derived. The roles are grouped to create agents which provide a set of services and the communication protocols identified during the interaction models will define the acquaintance model (see below) between the agents. The objective of each of these activities is summarised below.

- Roles model: Identify the different roles that exist in the system. A role describes what an entity is expected to do. The roles are characterised by their permissions and responsibilities.
- Interaction model: This model captures the relationship between the roles, defining the protocols which describe the interactions between them, i.e., the communication purpose, initiator, responder, inputs and outputs and processing information.
- Agent model: The agent model is created by aggregating roles into different agent entities.
- Services model: The services that each agent provides are defined based on the functions the agents provide, which are in turn derived from their associated roles and protocols.
- Acquaintance model: This model simply describes the communications links between the agents.

1.3 Other design issues

Gaia will allow us to specify and describe the ABM system (the agents, their services and relationships). However, it does not prescribe a specific simulator engine. Different approaches were considered (e.g., step-wise modelling, event-driven approach). The approach selected and its implications on the design of the model will be described in this deliverable.

There are different approaches to the design and implementation of the communication channels used in the model (e.g., with the use of a shared ontology and standardised communication messages, such as with a FIPA-compatible communication protocol [4], with the use of a communication middle server or with direct communication between agents in a closer object-oriented modelling approach). Once again, Gaia does not deal with this level of detail as this is related to the actual implementation of the system. These issues will also be considered separately.

Finally, note that these design and implementation issues affect the specification and design part, as some specific roles and messages might be needed. Therefore, we have followed an iterative process between the specification and design of the ABM system and the design of other parts of the model (simulation engine and communication) even if in this document, for clarity, these issues are not presented until Section 5.

1.4 Implementation details

In this deliverable, a set of implementation decisions will also be considered, such as the programming language, database description, the possibility of using off-the-shelf system development architectures, or the use of a software repository. All of these considerations will help us during the implementation phase of the project ensuring a smooth and coordinated development of the model. Again, note that some stress test have already been performed on different technological solutions which have an impact on the design phase of the ABM model.

1.5 Model description summary

Grimm et al. [5,6] propose a standardised protocol to describe agent-based systems called the 'overview, design concepts and details' (ODD) protocol. The goal of this protocol is to summarise in a pre-defined manner an ABM system so that other members in the research community could understand its main features to be able to reproduce the results. By its nature, the ODD is a protocol to describe an ABM model rather than a methodology to capture the specification and develop the design of a system. As we are in the phase of designing the model and changes might still occur to the system final implementation, we decided to use the Gaia methodology. ODD could be used in the future to summarise the system when reporting it in future publications.

1.6 Structure and contents of this deliverable

The document is structured following the Gaia methodology. We start presenting the requirements of the model (Section 2), followed by the analysis of the model to produce the specification of the ABM system described in Section 3. This section includes the roles and interactions models.

The design of the system using the previous analysis is detailed in Section 4. This includes the agent and service models and the acquaintance diagram.

As previously described, there are other implementation issues linked to the fact that we need a running simulator of the ATM. These issues (simulation engine, communication channels and parallelisation) are described in Section 5.

Section 6 is devoted at the discussion of other implementation practical details such as programming language, code repository and database modelling.

Section 7 contains a summary of next steps and the document closes with references and acronyms in Section 8 and 9 respectively.

The deliverable finishes with three annexes: Annex 1 with the full roles model, Annex 2 with the interactions model and Annex 3 with the services model.

2 System requirements

The requirements for the Domino model arise from two types of consideration: the effects we want to capture (e.g., degree of propagation of delay and cost of the elements in the system, identification of critical nodes in terms of delay and cost propagation) and what changes to the system we want to see the impact of (i.e., the impact of applying different mechanisms on key indicators for different stakeholders). Concerning the first point, Domino is mostly interested in how different processes interact. The model thus needs to include, at least:

- arrival management;
- departure management;
- pre-departure regulations;
- connecting passengers;
- aircraft turnaround.

In particular, we are aiming to capture how delay and costs propagate and how agents react to them. This includes the formation and resolution of congestion processes (runway, holding, ATFM regulations, etc). We are also interested in understanding how the anticipation of the disruptions by the agents shape the system. To do this, the model needs to consider different cost models and objective functions for the agents. One needs to include:

- the cost of delay for the airline including explicit consideration of passenger itineraries and their disruptions (missed connections and/or reaccommodation);
- the cost of fuel for the airline;
- network-wide impacts of cost propagation, considering reactionary delay (knock-on) effects which can be considered when optimising the airline flight plans;
- objective functions for departure and arrival managers and for the assignment of delay due to capacity constraints.

Note that the different elements in the ATM system that were identified in D3.1 (Architecture definition), will have to be included in the model, such as DMAN, E-AMAN, and Network Manager [1]. Concerning the mechanisms, our primary goal is to make predictions about the system and its key indicators when they are modified. The different mechanisms considered are detailed in Deliverable D3.1 (Architecture definition), but as a reminder, these mechanisms are:

- 4D trajectory adjustments: by changing its time of departure, its speed, or its trajectory, a flight is able to avoid some extra costs in disrupted operations.
- Flight prioritisation: by setting levels of priority for its flight, the airline is able to protect its most vital flights during periods of disruption.
- Flight arrival coordination: by extending the horizon and the capabilities of the arrival manager to deal with tactical capacity management of airport throughputs.

The implementation of each of the mechanisms creates some requirements for the model as a whole.

For the 4D trajectory adjustment mechanism, the model should be able to capture the effect of flights adjusting their planned 4D trajectory, using in particular speed adjustments (dynamic cost indexing), actively delaying on ground flights and using flight level and route changes. Note that in Domino we are interested in the knock-on effects of these changes, therefore, we'll focus on the time/cost that can be gained/lost with/without this mechanism. It might be required to do modifications of the trajectory at several points during the flight operation. For this reason, there is a need for the agents to have updated information during the flight to be able to make decisions based on it. Changes in trajectories might produce changes in demand which could affect the need of changes to the pre-tactically defined delay (e.g., ATFM delay). This could have an impact on strategic factors such as ANSPs investments which are out of scope of Domino.

The flight prioritisation process requires aircraft operators to predict the state of the network to be able to decide the prioritisation of their flights (i.e., measure the importance of their flights) and a centralised model which considers several flights at the same time to realise the prioritisation and delay assignment.

For the flight arrival coordination, the model needs to be able to build a queue for the arrival, with a parametrised horizon and a prioritisation mechanism to consider airlines requirements on the merging and sequencing of the arrivals. In the most advanced version of this mechanism, the arrival manager will need to communicate with the airline or another centralised body to get preferences and take them into account when building the arrival queue.

Finally, other non-functional requirements should be considered such as the scalability of the model (i.e., easiness of expanding and evolving the model) and the computational time required for a simulation run.

3 System analysis

3.1 Roles

3.1.1 Methodology

Following the Gaia methodology [11], a role can be seen as an abstract description of an entity's expected function. A role is defined by four attributes:

- Responsibilities: these determine functionality. They are divided between:
    - Liveness properties, which describe the states that the agent must bring about, given certain environmental conditions. For example, the different tasks that an agent needs to perform when a given environmental condition are met or a given message is received;
    - Safety properties, which are invariants that must be assured by the role.
- Permissions: rights associated with a role, which describe the resources that are available to the role to realise its responsibilities (e.g., reading, modifying or creating data)
- Activities: computations associated with the role that can be carried out without interacting with other agents, which can be seen as private actions in the context of software engineering.
- Protocols: these define the way that a role can interact with other roles.

As explained in Section 1.2, even if the roles do not change for a given scenario, different scenarios in Domino model different descriptions of the mechanisms which might affect how some of the roles behave. For this reason, when the level of implementation of the mechanism has an impact on the roles this will be indicated as described in Table 1.

Table 1. Mechanisms implemented and their associated code

Mechanism code	Name	Description
4DT_0	4D Trajectory adjustment at Level 0	Basic delay recovery rules linked to amount of delay incurred; cost index computed before departure; limited waiting for passengers; basic re-routing and level changes used.
4DT_1	4D Trajectory adjustment at Level 1	Use of dynamic cost indexing for flights, with a simplified in-flight delay and cost estimation based on heuristics and general cost of delay rules; wait for passengers used more widely but with limited rules only; estimation of possibility of re-routing and level changes with respect to

		dynamic cost indexing
4DT_2	4D Trajectory adjustment at Level 2	Advanced estimation of expected costs due to connecting delayed flights and optimisation of actions considering the three available techniques (dynamic cost indexing, trajectory modification and wait-for-passenger decisions).
FAC_0	Flight arrival coordination at Level 0	Current principles applied on E-AMAN systems; the flight arrival coordinator tries to minimise the amount of holding delay that will be carried out at the TMA by providing speed advisories for flights during their en-route phase; the AMAN in the TMA is focused on the maximisation of throughput at the runway; no information from the airlines is taken into account when applying this mechanism.
FAC_1	Flight arrival coordination at Level 1	The flight arrival coordinator tries to minimise the expected reactionary ('knock-on') delay, considering information from the airport on the expected turnaround of flights.
FAC_2	Flight arrival coordination at Level 2	The airspace user provides the prioritisation of their flights when entering the arrival coordinator's planning horizon or the priorities are defined pre-tactically (i.e. flight prioritisation (e.g. UDPP) choices are relayed to the flight arrival coordinator and taken into account when creating the landing sequence).
FP_0	Flight prioritisation at Level 0	No pre-departure prioritisation of flights by the AU, use of first-planned first-served (FPFS) when assigning ATFM delay; no slot swapping takes place
FP_1	Flight prioritisation at Level 1	Pre-departure prioritisation of flights, allowing AUs to reorder several flights in the same constraint among their own slots when they are delayed (UDPP principles); includes Enhanced Slot Swapping (SESAR1)
FP_2	Flight prioritisation at Level 2	As per FP_1, plus a 'credit' system so that airlines can prioritise flights at a given airport using credits previously earned at the same airport.

The roles will be described following a role schema as shown in Table *2*.

Table 2. Role schema description

Role schema	Name of role
Description	Short verbal / colloquial description of the role
Protocol and activities	*Protocols* (shown in italics) and activities (shown underlined) in which the role plays a part
Permissions	Rights associated with the role
Liveness responsibilities	Liveness responsibilities

Safety responsibilities	Safety responsibilities

3.1.2 Roles in Domino

There are a total of 38 roles identified in Domino. For readability of the document, here only their name and description are presented. The full role model is compiled in Annex 1.

Table 3. Role schema description Airline Flight Planner (AFP)

Role schema	Airline Flight Planner (AFP)
Description	Select which flight plan will be executed. This means selecting the 4D trajectory (including delayed departing time (e.g., waiting for passengers)) and possibly cancelling flight (i.e., the Airline Flight Planner decides to cancel the flight plan). 1. Check which flights are ready for pre-departure (less than 3 hours before their EOBT and in scheduled status) or get a request to recompute the flight plan selection of a given flight. 2. Submit flight plan which might lead to ATFM delay update. 3. Check different flight plan options and their costs. 4. Decide which flight plan to execute tactically. The liveness of AFP will depend on the mechanism 4DT and the FP implementation. The action of RequestATFMSlot will depend on the FP mechanism: • FP_0: Request ATFM slot • FP_1 and FP_2: Request ATFM slot and then prioritise the flight and optionally request a flight swap. The action of DecideOption depends on 4DT: • 4DT_0: Rule of thumb to decide between speed-up, wait-for-passengers and cancellation • 4DT_1 and 4DT_2: Selection of option based on cost provided by CheckFPOption
Full description	Annex 1: Table 47. Role schema description Airline Flight Planner (AFP)

Table 4. Role schema description Network Manager Flight Plan Processing (NMFPP)

Role schema	Network Manager Flight Plan Processing (NMFPP)
Description	Process a flight plan submission by the NM. It checks if ATFM delay is needed and if it is the case returns the ATFM delay.
Full description	Annex 1: Table 48. Role schema description Network Manager Flight Plan Processing (NMFPP)

Table 5. Role schema description Network Manager Accept and Disseminate FP (NMAD)

Role schema	Network Manager Accept and Disseminate FP (NMAD)
Description	Request the dissemination of the Flight Plan to the entities interested in it and returns the points where the Flight needs to notify when reaching them.
Full description	Annex 1: Table 49. Role schema description Network Manager Accept and Disseminate FP (NMAD)

Table 6. Role schema description Network Manager Cancel FP (NMC)

Role schema	Network Manger Cancel FP (NMC)
Description	Request the cancellation of a Flight Plan. Request the dissemination of the cancellation of a flight plan.
Full description	Annex 1: Table 50. Role schema description Network Manager Cancel FP (NMC)

Table 7. Role schema description Disseminate FP (DFP)

Role schema	Disseminate FP (DFP)
Description	Send the FP to the entities interested in it to get the points when it need to be notified (E-AMAN and DMAN)
Full description	Annex 1: Table 51. Role schema description Disseminate FP (DFP)

Table 8. Role schema description Disseminate Cancellation FP (DCFP)

Role schema	Disseminate Cancellation FP (DCFP)
Description	Send the information that a FP has been cancelled to the entities interested in so that its slots are released (E-AMAN and DMAN)
Full description	Annex 1: Table 52. Role schema description Disseminate Cancellation FP (DCFP)

Table 9. Role schema description Departure Queue Updater (DQU)

Role schema	Departure Queue Updater (DQU)
Description	When a flight updates its EOBT, its position in the departure queue is updated.
Full description	Annex 1: Table 53. Role schema description Departure Queue Updater (DQU)

Table 10. Role schema description Arrival Queue Planned Updater (AQPU)

Role schema	Arrival Queue Planned Updater (AQPU)
Description	Update the queue of flights planned to arrive with information from the AOC when a flight update its EIBT. If it is the first time the flight is provided then send the requests of points where the E-AMAN needs to be notified of the flight.
Full description	Annex 1: Table 54. Role schema description Arrival Queue Planned Updater (AQPU)

Table 11. Role schema description Passenger Reallocation (PR)

Role schema	Passenger Reallocation (PR)
Description	Decide how to manage connecting passengers when a flight has left. 1. Check passenger that should have been in the flight that has left and have missed their connection 2. Rebook them onto following flights to destination, compensate and return them to destination, pay for care, and potentially put them in hotels by checking preferences with passengers.
Full description	Annex 1: Table 55. Role schema description Passenger Reallocation (PR)

Table 12. Role schema description Passenger Preferences in Connection (PPC)

Role schema	Passenger Preferences in Connection (PPC)
Description	Provides the preferences of passengers for follow up connections when missed connection.
Full description	Annex 1: Table 56. Role schema description Passenger Preferences in Connection (PPC)

Table 13. Role schema description Provide Connecting Times (PCT)

Role schema	Provide Connecting Times (PCT)
Description	Provides connecting times for passengers at airport
Full description	Annex 1: Table 57. Role schema description Provide Connecting Times (PCT)

Table 14. Role schema description Airline Pre-departure Flight Prioritiser (ADFP)

Role schema	Airline Pre-departure Flight Prioritiser (ADFP)
Description	Compute the priority of a flight. Used in FP_1 and FP_2. ComputeFlightPriority in FP_2 considers also the use of the credit system.
Full description	Annex 1: Table 58. Role schema description Airline Pre-departure Flight Prioritiser (ADFP)

Table 15. Role schema description Flight Swapper (FS)

Role schema	Flight Swapper (FS)
Description	Swap flights with ATFM delay for flight prioritisation mechanisms. Behaviour dependent on level of FP mechanism: • FP_1: Use of priorities provided by flights. Allow swapping of flights within an airline. • FP_2: Use of priorities provided by flights. Allow swapping of flights between airlines.
Full description	Annex 1: Table 59. Role schema description Flight Swapper (FS)

Table 16. Role schema description Airline Pre-departure Flight Plan Cost Computer (AFPC)

Role schema	Airline Pre-departure Flight Plan Cost Computer (AFPC)
Description	Compute cost of changing the 4D trajectory of a given flight at pre-departure stage. Provides the best option and cost for each of the change possibilities: wait for passengers, modifying 4D trajectory, cancelling flight. The role is different depending on 4DT: • 4DT_1 and 4DT_2: Same liveness but different degree of depth of analysis, since 4DT_2 considers downstream effects.
Full description	Annex 1: Table 60. Role schema description Airline Pre-departure Flight Plan Cost Computer (AFPC)

Table 17. Role schema description Flight Plan Updater (FPU)

Role schema	Flight Plan Updater (FPU)
Description	When the flight plan is changed the information is updated for the flight. 1. Wait for request of update flight plan 2. Update information on flight plan including when to report 3. If the flight is airborne then request a flight parameters recomputation
Full description	Annex 1: Table 61. Role schema description Flight Plan Updater (FPU)

Table 18. Role schema description Wait for Pushback_Ready (WPBR)

Role schema	Wait for Pushback_Ready (WPBR)
Description	When the flight is ready for departure then wait for push back ready event
Full description	Annex 1: Table 62. Role schema description Wait for Pushback_Ready (WPBR)

Table 19. Role schema description Departure Slot Requester (DSR)

Role schema	Departure Slot Requester (DSR)
Description	When the time of the EOBT ready aircraft arrives then the departing slot is requested. 1. Request departure slot 2. Request estimate taxi-out time
Full description	Annex 1: Table 63. Role schema description Departure Slot Requester (DSR)

Table 20. Role schema description Taxi Out Estimator (TOE)

Role schema	Taxi Out Estimator (TOE)
Description	Provides an estimation of the taxi out time required for a flight
Full description	Annex 1: Table 64. Role schema description Taxi Out Estimator (TOE)

Table 21. Role schema description Departure Slot Provider (DSP)

Role schema	Departure Slot Provider (DSP)
Description	When a flight is ready at the gate, sends a WaitForDepartureRequest to get the slot in the runway sequence. The role gets the request with the CTOT (if any) and provides a departure slot in the runway sequence. The role takes into account the taxi-out times and the order of the flights in the departure queue.
Full description	Annex 1: Table 65. Role schema description Departure Slot Provider (DSP)

Table 22. Role schema description Aircraft Departing Handler (ADH)

Role schema	Aircraft Departing Handler (ADH)
Description	When the time of the push back arrives then the taxi out and the take-off time are computed. 1. Request taxi-out time 2. Compute takeoff time
Full description	Annex 1: Table 66. Role schema description Aircraft Departing Handler (ADH)

Table 23. Role schema description Taxi Out Provider (TOP)

Role schema	Taxi Out Provider (TOP)
Description	Provides the actual taxi out time for a flight
Full description	Annex 1: Table 67. Role schema description Taxi Out Provider (TOP)

Table 24. Role schema description Operate Trajectory (OT)

Role schema	Operate Trajectory (OT)
Description	Fly until the next notification Flight Cross Point or Landing.
Full description	Annex 1: Table 68. Role schema description Operate Trajectory (OT)

Table 25. Role schema description EIBT Updater (EU)

Role schema	EIBT Updater (EU)
Description	Update the EIBT of a flight for the AO
Full description	Annex 1: Table 69. Role schema description EIBT Updater (EU)

Table 26. Role schema description Strategic Departure Queue Builder (SDQB)

Role schema	Strategic Departure Queue Builder (SDQB)
Description	Builds the departure queue at an airport based on the flight schedules and airport departure capacity
Full description	Annex 1: Table 70. Role schema description Strategic Departure Queue Builder (SDQB)

Table 27. Role schema description Strategic Arrival Queue Builder (SAQB)

Role schema	Strategic Arrival Queue Builder (SAQB)
Description	Builds the arrival queue at an airport based on the flight schedules and airport arrival capacity
Full description	Annex 1: Table 71. Role schema description Strategic Arrival Queue Builder (SAQB)

Table 28. Role schema description Ground Arrival Handler (GAH)

Role schema	Ground Arrival Handler (GAH)
Description	When a flight has landed, this role computes when it will arrive at the gate. 1. Compute the taxi-in time. 2. Update AIBT with final time
Full description	Annex 1: Table 72. Role schema description Ground Arrival Handler (GAH)

Table 29. Role schema description Taxi In Provider (TIP)

Role schema	Taxi In Provider (TIP)
Description	Provides the actual taxi in time for a flight
Full description	Annex 1: Table 73. Role schema description Taxi In Provider (TIP)

Table 30. Role schema description Turnaround Operations (TRO)

Role schema	Turnaround Operations (TRO)
Description	When a flight arrives to the gate computes the turnaround time required, generates the ready to depart time of the subsequent flight. 1. Reallocate passengers of arriving flight if needed 2. Computes the turnaround time. 3. Computes delay due to non-ATFM causes 4. Updates the EOBT 5. Requests reassessment of flight departure in case extra delay has been incurred
Full description	Annex 1: Table 74. Role schema description Turnaround Operations (TRO)

Table 31. Role schema description Airline Passenger Handler (APH)

Role schema	Airline Passenger Handler (APH)
Description	When a flight arrives direct passengers to connecting flights. All passengers connecting to flight that have already left are requested to be rebooked.
Full description	Annex 1: Table 75. Role schema description Airline Passenger Handler (APH)

Table 32. Role schema description Ground Handler (GH)

Role schema	Ground Handler (GH)
Description	Provides the turnaround time for a flight
Full description	Annex 1: Table 76. Role schema description Ground Handler (GH)

Table 33. Role schema description Departure Reassessment (DR)

Role schema	Departure Reassessment (DR)

Description	When the EOBT of a flight is updated we might have to reassess the ATFM delay and the flight plan of the flight if the flight was already in pre-departure status. 1. If the flight has an ATFM slot and this is missed: request a new ATFM slot and update EOBT. 2. If flight plan could be modified call to RequestFlightPlanRecomputation
Full description	Annex 1: Table 77. Role schema description Departure Reassessment (DR)

Table 34. Role schema description Disseminate Flight Position Update (DFPU)

Role schema	Disseminate Flight Position Update (DFPU)
Description	Updates subscribed entities with position report for flight
Full description	Annex 1: Table 78. Role schema description Disseminate Flight Position Update (DFPU)

Table 35. Role schema description Flight in AMAN Handler (FIAH)

Role schema	Flight in AMAN Handler (FIAH)
Description	Get notified that a flight has entered/moved in the AMAN and notify the required service from the AMAN
Full description	Annex 1: Table 79. Role schema description Flight in AMAN Handler (FIAH)

Table 36. Role schema description Arrival Queue Updater (AQU)

Role schema	Arrival Queue Updater (AQU)
Description	When a flight enters the planning scope of the E-AMAN, its position in the arrival queue is updated. The way the queue is updated is dependent on the FAC implementation: • FAC_0: 'First-in, first-out' approach. First slot available is given to the aircraft. • FAC_1: Reactionary delay is considered when sequencing flights. An estimation of the reactionary delay that will be generated is computed and used in the prioritisation process. • FAC_2: Priorities set by airspace users are considered A message is sent to the flight to update the arrival delay expected.
Full description	Annex 1: Table 80. Role schema description Arrival Queue Updater (AQU)

Table 37. Role schema description Flight Arrival Information Provider (FAIP)

Role schema	Flight Arrival Information Provider (FAIP)
Description	Provides information needed to consider the arrival sequencing. • FAC_1: The information provided is related to the reactionary delay • FAC_2: The information provided are the priorities set by airspace users Process might request information from the AOC.
Full description	Annex 1: Table 81. Role schema description Flight Arrival Information Provider (FAIP)

Table 38. Role schema description Flight Arrival Information Estimator (FAIE)

Role schema	Flight Arrival Information Estimator (FAIE)
Description	Provides information needed to consider the arrival sequencing from the AOC perspective. • FAC_1: The information provided is related to the reactionary delay • FAC_2: The information provided are the priorities set by airspace user
Full description	Annex 1: Table 82. Role schema description Flight Arrival Information Estimator (FAIE)

Table 39. Role schema description Arrival Slot Provider (ASP)

Role schema	Arrival Slot Provider (ASP)
Description	When a flight enters the execution horizon the slot is assigned to the flight, thus fixing the arrival queue.
Full description	Annex 1: Table 83. Role schema description Arrival Slot Provider (ASP)

Table 40. Role schema description Flight Plan Constraint Updater (FPCU)

Role schema	Flight Plan Constraint Updater (FPCU)
Description	Updates constraints of flight plan of a flight.
Full description	Annex 1: Table 84. Role schema description Flight Plan Constraint Updater (FPCU)

3.2 Interactions

3.2.1 Methodology

The links between roles are represented in Gaia in the interaction model. This model consists of a set of protocol definitions. The focus is on the purpose of the interaction, rather than on the precise ordering of particular message exchanges. A protocol definition consists of the following attributes:

- purpose: brief textual description of the nature of the interaction;
- initiator: the role(s) responsible for starting the interaction;
- responder: the role(s) with which the initiator interacts;
- inputs: information used by the role initiator while enacting the protocol;
- outputs: information supplied by/to the protocol responder during the course of the interaction;
- processing: brief textual description of any processing the protocol initiator performs during the course of the interaction.

These elements will be arranged for each interaction as shown in Table 41. As in the role model, when differences in the interaction model due to the level of the mechanisms exists, they will be indicated explicitly

Table 41. Interaction model description

Purpose	
Initiator	Responder
Inputs	
Processing	
Outputs	

3.2.2 Interaction model in Domino

In order to help the understanding of the connections between the different roles via the different protocols, a graphical representation of the communications between them is shown in Figure 2.

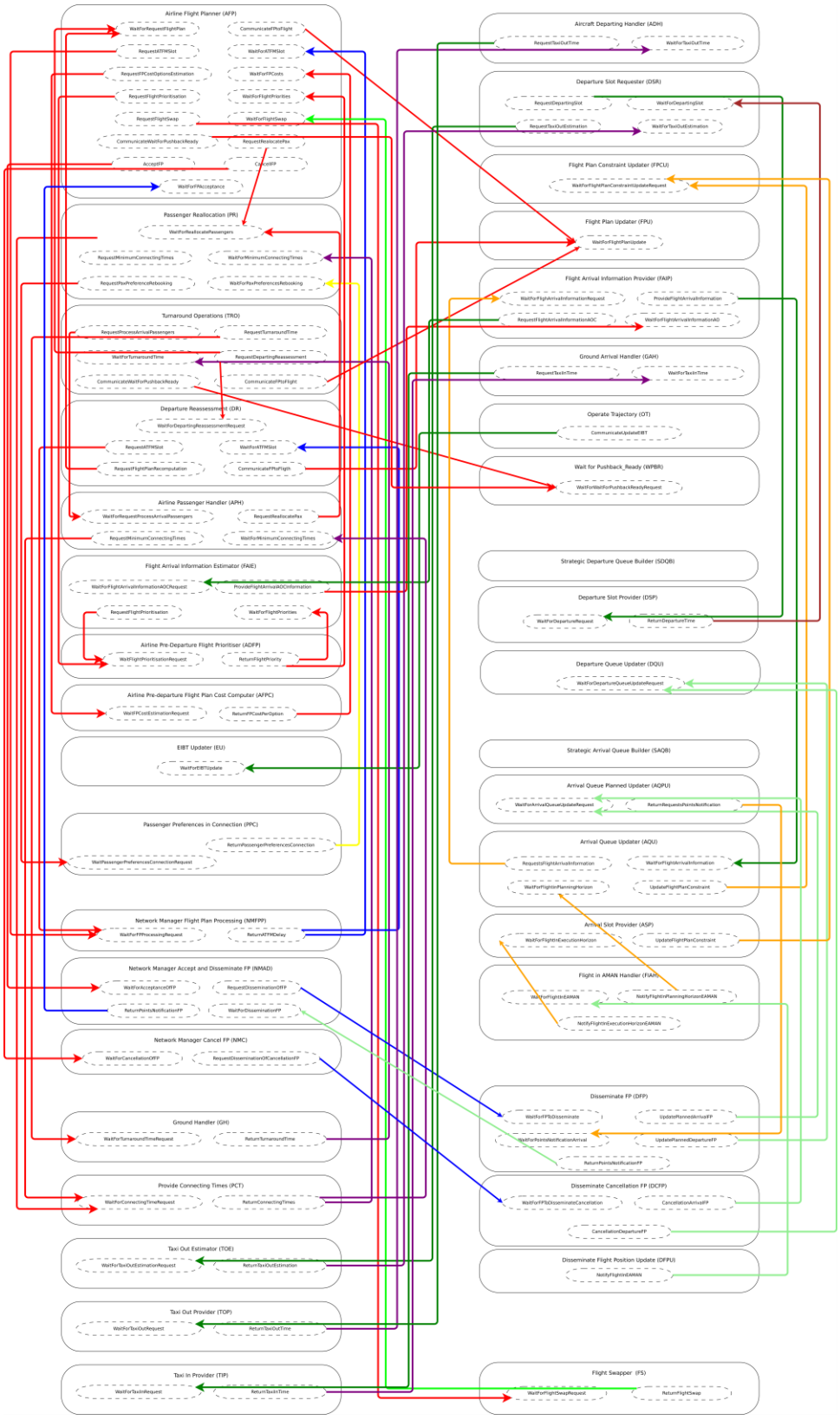

Figure 2: Roles in Domino and their protocols connections

The interactions are fully described following the Gaia methodology grouped by the role that starts them in Annex 2. Table 42 summarises the list of the different interaction model descriptions compiled in the Annex.

Table 42. Interaction model descriptions in Annex 2

Role starts interaction	High level description of interaction	Interaction models
Airline Flight Planner	Request ATFM slot	Table 85. Interaction model Request ATFM slot Table 86. Interaction model Return ATFM delay
	Request cost options of FP modifications	Table 87. Interaction model Request FP Cost Options Estimation Table 88. Interaction model Return FP Cost per Option
	Request flight piroritisation	Table 89. Interaction model Request Flight Prioritisation Table 90. Interaction model Return Flight Priority
	Request flight swap	Table 91. Interaction model Flight Swap Request Table 92. Interaction model Return Flight Swap
	Accept flight plan	Table 93. Interaction model Accept FP Table 94. Interaction model Return Point Notification FP Table 95. Interaction model Request Dissemination of FP Table 96. Interaction model Update Planned Arrival of FP Table 97. Interaction model Return Request Points Notification Table 98. Interaction model Update Planned Departure FP Table 99. Interaction model Communicate FP to Flight
	Cancel flight plan	Table 100. Interaction model Communicate FP to Flight Table 101. Interaction model Cancel FP Table 102. Interaction model Request Dissemination Cancellation FP Table 103. Interaction model Cancellation Arrival FP Table 104. Interaction model Cancellation Departure FP
	Communicate flight ready to wait for push back	Table 105. Interaction model Communicate Wait for Pushback Ready

	Request reallocation of passengers	Table 106. Interaction model Passenger Reallocation Request Table 107. Interaction model Request Minimum Connecting Times Table 108. Interaction model Return Connecting Times Table 109. Interaction model Request Passengers Preference Rebooking Table 110. Interaction model Return Passenger Preferences in Connection
Airline Turnaround Operations	Request reassessment of departing	Table 111. Interaction model Request Departing Reassessment
	Request process arrival passengers	Table 112. Interaction model Request Process Arrival Passengers Table 113. Interaction model Request Minimum Connecting Times Table 114. Interaction model Return Connecting Times Table 115. Interaction model Request Reallocate Passengers
	Aircraft turnaround operations	Table 116. Interaction model Request Turnaround Time Table 117. Interaction model Return Turnaround Time Table 118. Interaction model Communicate FP to Flight Table 119. Interaction model Communicate Wait For Pushback Ready
Departure Reassessment	Request ATFM slot	Table 120. Interaction model Request ATFM Slot Table 121. Interaction model Return ATFM Slot
	Request recomputation of flight plan	Table 122. Interaction model Request Flight Plan Recomputation
	Communicate flight plan update	Table 123. Interaction model Communicate FP to Flight
Flight Arrival Information Estimator	Request flight prioritization	Table 124. Interaction model Request Flight Prioritisation Table 125. Interaction model Return Flight Prioritisation
Aircraft Departing Handler	Request taxi out time	Table 126. Interaction model Request Taxi Out Time Table 127. Interaction model Return Taxi Out Time

Departing Slot Requester	Request taxi out estimation	Table 128. Interaction model Request Taxi Out Time Estimation Table 129. Interaction model Return Taxi Out Time Estimation
	Request departing slot	Table 130. Interaction model Request Departing Slot Table 131. Interaction model Return Departure Time
Ground Arrival Handler	Request taxi in time	Table 132. Interaction model Request Taxi In Time Table 133. Interaction model Return Taxi In Time
Operate Trajectory	Communicate update EIBT	Table 134. Interaction model Communicate Update EIBT
Disseminate Flight Position Update	Disseminate flight has moved within the E-AMAN	Table 135. Interaction model Notify Flight In EAMAN
Flight in AMAN Handler	Notify E-AMAN flight has entered Planning Horizon	Table 136. Interaction model Notify Flight In Planning Horizon
	Notify E-AMAN flight has entered Execution Horizon	Table 137. Interaction model Notify Flight In Execution Horizon
Arrival Queue Updater	Request flight arrival information to update arrival slot	Table 138. Interaction model Request Flight Arrival Information Table 139. Interaction model Request Flight Arrival Information AOC Table 140. Interaction model Provide Flight Arrival AOC Information Table 141. Interaction model Provide Flight Arrival Information
	Provide updated arrival slot to flight	Table 142. Interaction model Update Flight Plan Constraint
Arrival Slot Provider	Provide arrival slot to flight	Table 143. Interaction model Update Flight Plan Constraint

4 Model design

4.1 Agent model

4.1.1 Methodology

The objective of the Gaia agent model is to document the various agent types that will be used in the system under development, and the agent instances that will realise these agent types at run-time. An agent type is best thought of as a set of agent roles. There may in fact be a one-to-one correspondence between roles and agents, but this is not mandatory. The agent model is defined using a simple agent type tree, in which leaf nodes correspond to roles (as defined in the roles model), and other nodes correspond to agent types. The agent instances that will appear in a system are annotated in the agent model.

4.1.2 Agent model in Domino

There are a total of 9 agent types modelled in Domino which group the 38 roles identified in the system analysis as follows (note that below, a '+' indicates that there will be multiple instances of this agent, whereas '1' indicates that there will be only 1):

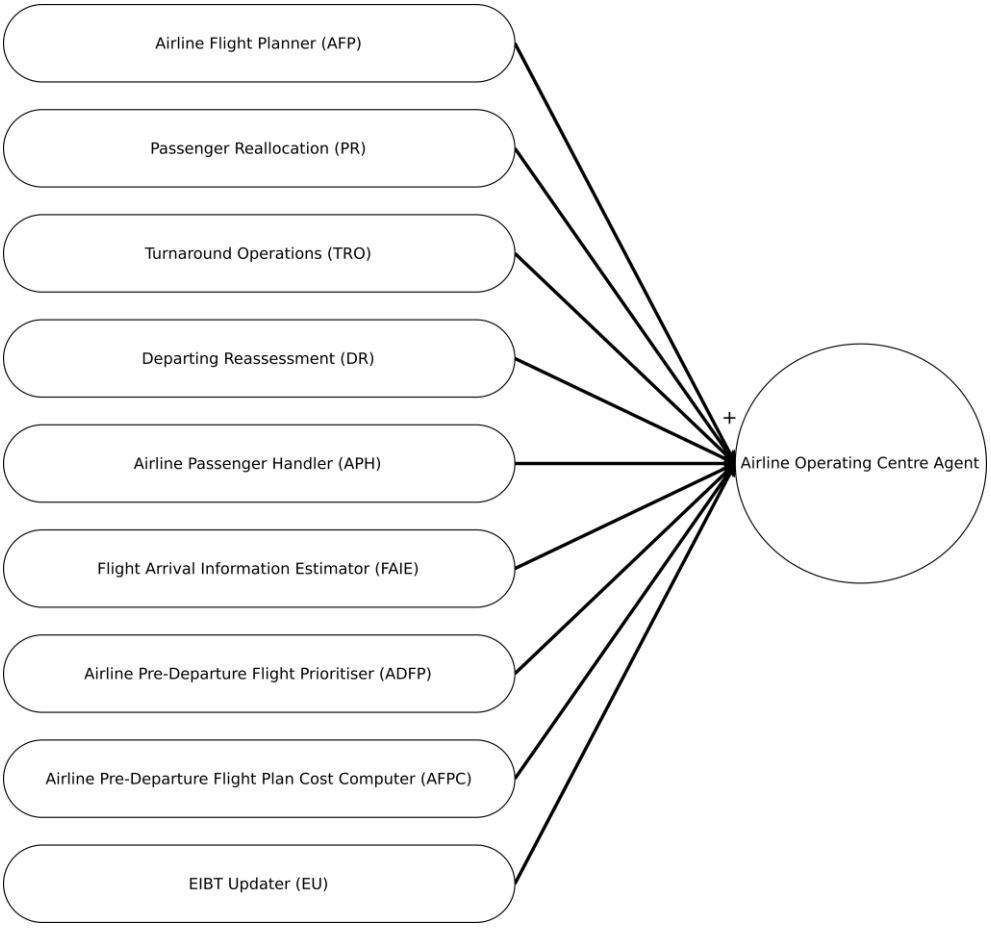

Figure 3: Agent model Airline Operating Centre

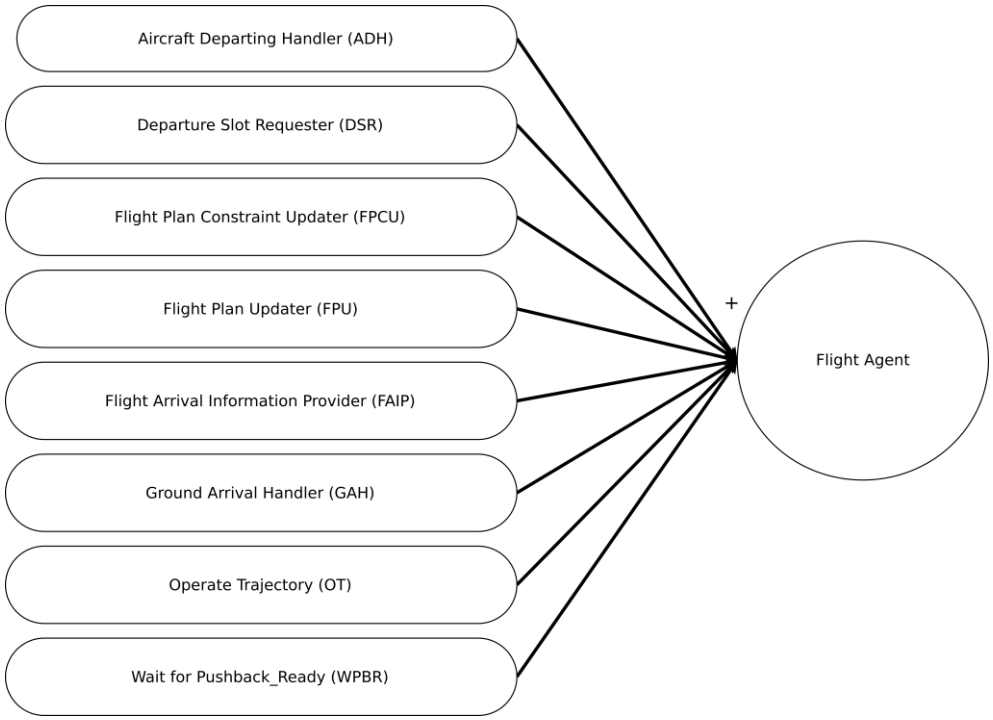

Figure 4: Agent model Flight

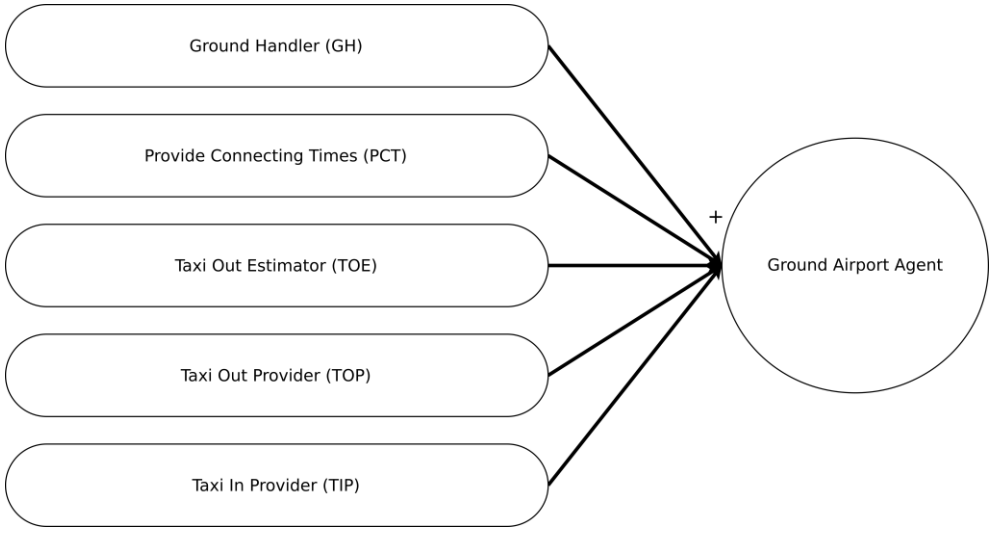

Figure 5: Agent model Ground Airport

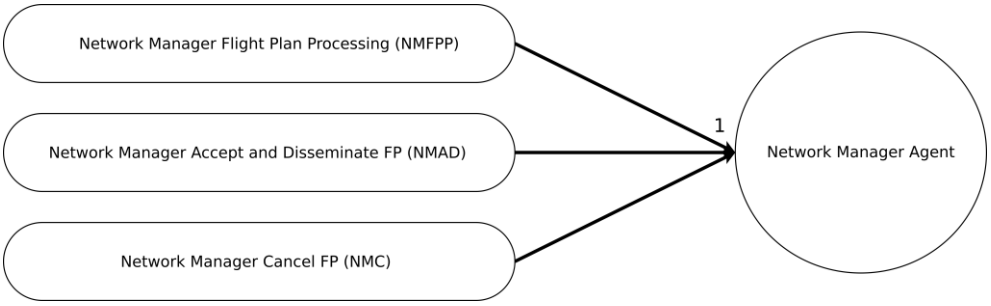

Figure 6: Agent model Network Manager

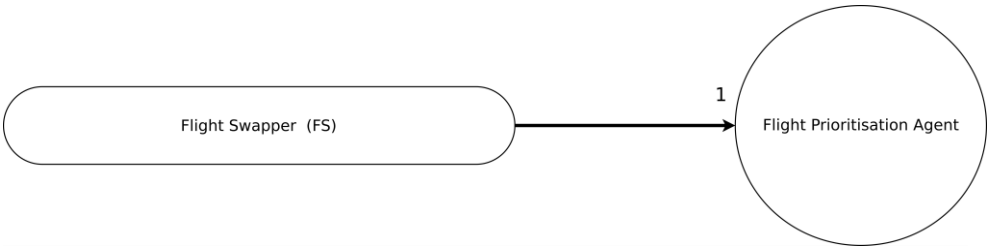

Figure 7: Agent model Flight Prioritisation

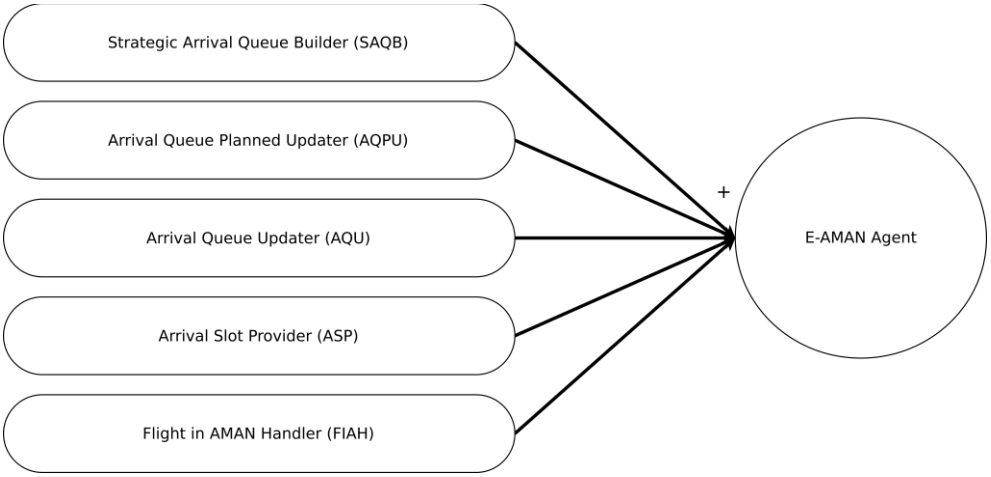

Figure 8: Agent model E-AMAN

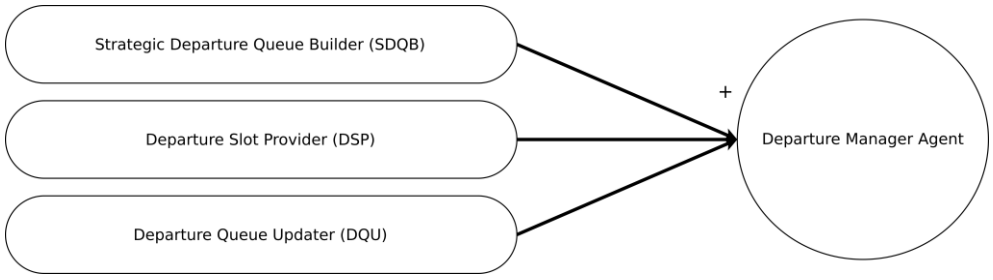

Figure 9: Agent model DMAN

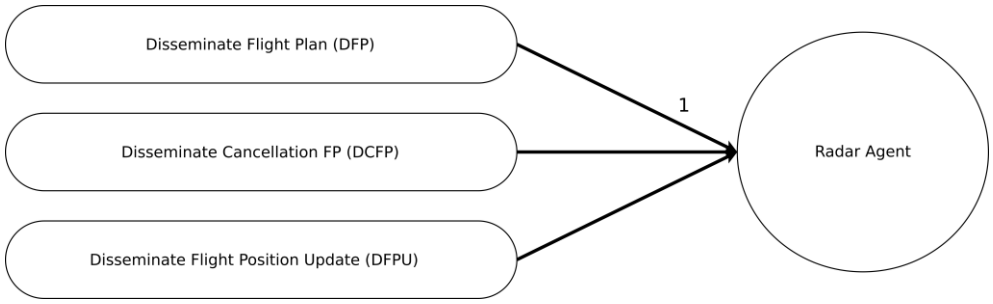

Figure 10: Agent model Radar

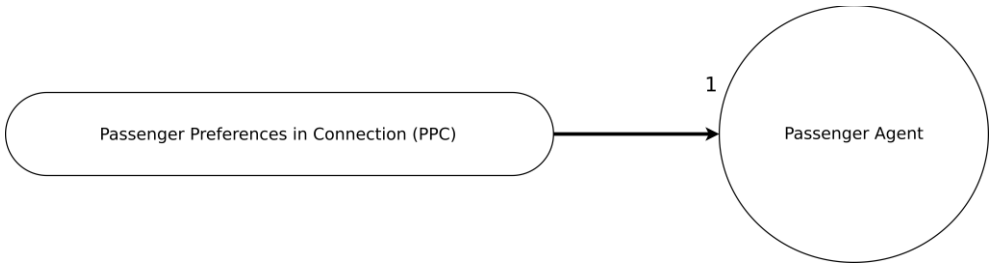

Figure 11: Agent model Passenger

4.2 Services model

4.2.1 Methodology

The aim of the Gaia services model is to identify the services associated with each agent role, and to specify the main properties of these services. A service is a function of the agent. A service is simply a single, coherent block of activity in which an agent will engage. Each service will have a set of properties defined:

- inputs: derived from the protocols of the roles that form the agent;
- outputs: derived from the protocols of these roles;
- pre-conditions: constraints on services derived from the safety properties of these roles;

- post-conditions: constraints on services derived from the safety properties of these roles.

The services will be grouped per agent and described as shown in Table 43.

Table 43. Service model description

Service	Inputs	Outputs	Pre-condition	Post-condition
Service name	Inputs that the service needs Note, we mark with * the inputs that are from events in the model	Outputs that the service generates	Pre-conditions that need to be meet before the service is provided	Post-conditions that are ensured after the service is executed

4.2.2 Services model in Domino

The services defined per agent are described in Table 44. Annex 3 contains the full description of the services model in Domino as summarised in Table 45.

Table 44. Services defined per agent

Agent	Service
Airline Operating Centre	Check flights are ready for flight plan submission
	Check flight has pushed back
	Check flight has arrived
	Update flight plan information
	Request ATFM slot
	Compute flight plan options cost estimation
	Choose flight plan option
	Request flight prioritisation
	Request flight swap
	Recompute EOBT
	Recompute EIBT
	Accept flight plan
	Cancel flight plan
	Communicate wait for pushback to flight
	Update EOBT from EIBT change
	Obtain following flight

	Departure reassessment
	Compute passenger missing connections
	Estimate information for arrival coordination of a flight
	Request connecting times for passengers
	Request priorities for passengers rebooking
	Check load factors of flights
	Compute turnaround of flight
	Request turnaround time for flight
Flight	Check push back ready event arrives
	Check push back event arrives
	Check Takeoff event arrives
	Check Flight crosses significant point
	Check flight has landed
	Request taxi in time
	Request taxi out estimation
	Request taxi out time
	Compute flight segment
	Compute delay non due to ATFM
	Update flight plan information
	Update flight plan constraint
	Request flight arrival information from AOC
	Provide flight arrival information
	Wait for pushback ready
	Communicate EIBT update
	Update EIBT
	Compute take off time
	Compute AOBT
	Request departing slot
Radar	Check Flight crosses significant point
	Disseminate flight crossing point update
	Disseminate flight plan
	Disseminate cancellation of flight plan

Network Manager	Flight plan processing
	Accept and disseminate flight plan
	Cancel flight plan
Ground Airport Agent	Provide connecting times
	Provide taxi out estimation
	Provide taxi out
	Provide taxi in
	Provide turnaround time
Flight Prioritisation	Compute flight swap
Passenger	Provide passengers preferences in connection
E-AMAN	Build arrival queue
	Update planned arrival queue
	Request arrival information to flight
	Update arrival queue
	Communicate arrival slot to flight with updated flight plan constraint
	Handle flight in EMAN
DMAN	Build departure queue
	Update planned departure queue
	Provide departing slot

Table 45. Service model descriptions in Annex 3

Agent	Service model
Airline Operating Centre	Table 144. Service model for Airline Operating Centre Agent
Flight	Table 145. Service model for Flight Agent
Radar	Table 146. Service model for Radar Agent
Network Manger	Table 147. Service model for Network Manager Agent
Ground Airport	Table 148. Service model for Ground Airport Agent
Flight Prioritisation	Table 149. Service model for Flight Prioritisation Agent
Passenger	Table 150. Service model for Passenger Agent
E-AMAN	Table 151. Service model for E-AMAN Agent
DMAN	Table 152. Service model for DMAN Agent

4.3 Acquaintances model

4.3.1 Methodology

The acquaintances model simply defines the communication links that exist between agent types. They do not define what messages are sent or when messages are sent — they simply indicate that communication pathways exist. In particular, the purpose of an acquaintances model is to identify any potential communication bottlenecks, which may cause problems at runtime. The acquaintance model may be derived in a straightforward way from the roles, protocols, and agent models.

4.3.2 Acquaintances model in Domino

In order to facilitate the identification of which protocols are internal within the agents and which ones require communication between them, the different protocols identified in the roles have been grouped per agent as shown in Figure 12.

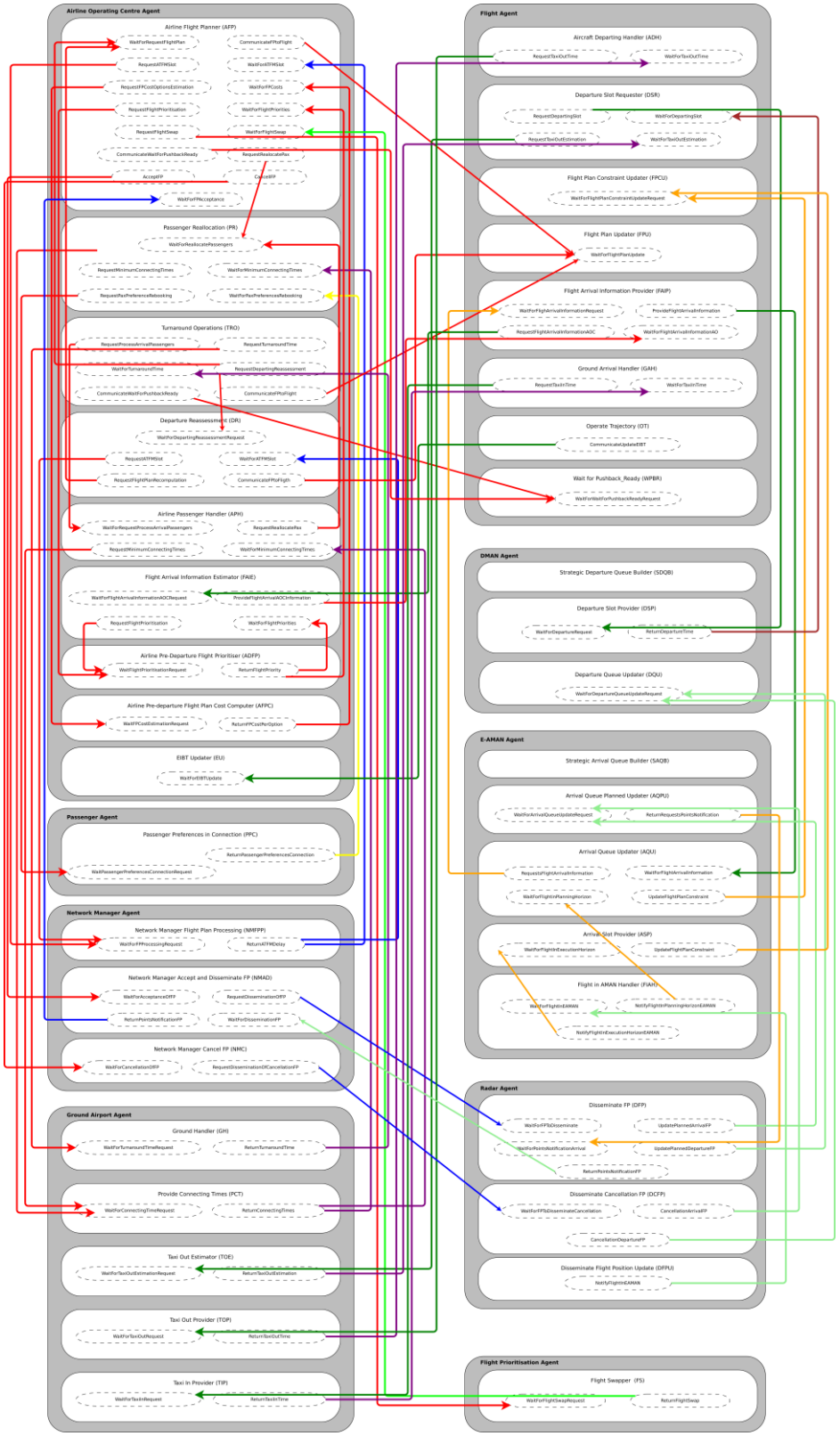

Figure 12: Roles and their protocols with Agents

From the analysis of Figure 12 we can identify the acquaintance of the agents as shown in Figure 13.

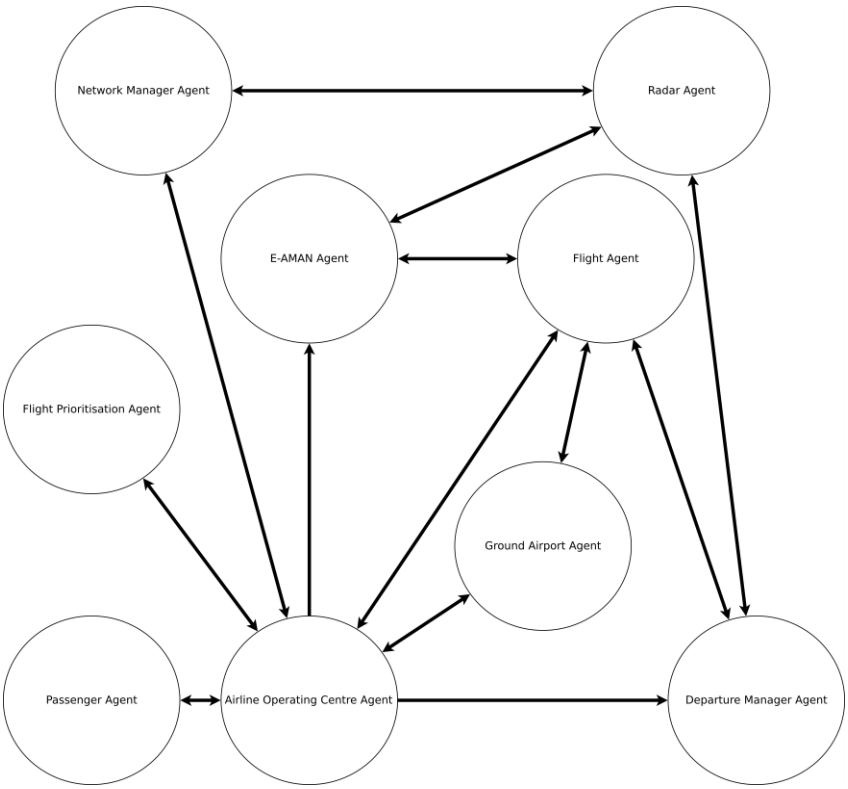

Figure 13: Agent acquaintance model

The AOC agent is a central one which communicates with most of the agents as it is in charge of the airline operations and decisions that need to be performed in terms of flight dispatching and passenger itineraries management

5 Other design issues

5.1 Simulation engine

As described in [9], a pure ABM system could be considered to be one where each agent has its own thread of control (being decentralised) and they are active entities, i.e., entities that can take on the initiative to do something. There is no concept of flow as understood in a sequential programming approach as the agents are driven by their reactions to changes in the environment. Therefore, when developing an agent-based model, one describes the behaviour of the different agents and their interactions, but the evolution of the environment is needed in order to advance the simulation. As described in [8], there are different possibilities to perform this system evolution. Notably, agents could operate in a:

- step-driven simulator, where actions are performed in turns by the agents,
- in a time-driven environment, where actions occur continually at fixed time steps,
- in a discrete-event simulation, which is event-driven and actions at performed only when an event occur,
- or in a message-based computation environment, where actions are taken when a message is received by an agent [10].

One of the main advantages of an event-driven approach is that the runs of the model can be done faster as the simulator can directly jump to the time stamp of the first event in the queue of events [8]. Notably, event-driven approach could be more suitable than a time-driven environment as it is more efficient, can improve the accuracy/validity/coherency of the results as it limits the number of actions that need to be performed simultaneously (i.e, within a given time step), and improves the congruence as agents don't need to be notified of actions simultaneously [8].

For these reasons, an event-driven simulation engine is considered in Domino. This paradigm is suited for Domino as it allows us to model the evolution of the system with the level of detail required without having to track the flights on a time-continuous environment. Some platforms such as MASON allow to schedule processes and events on their built-in scheduler [7]. However, as acknowledged in [9], there are no established frameworks or methodologies to guide researchers and analysts through the agent-based modelling and simulation process.

In our case, the different events will act as triggers to actions and interactions performed by the agents in the model. An event will trigger a set of interactions between agents and once all these interactions are finished, the next event will be considered. The actions that the agents perform will in their turn create future events that will be added to the queue of events. As mentioned before, the times between actions are not fixed as the simulator directly jumps to the time when the next

event in the queue happens. Therefore, the number and type of events need to be defined. When a given event has occurred, the agents need to be able to react to it.

5.1.1 Event driven and ABM in Domino

The approach considered in Domino is that the agents have visibility of the current event, but only the owners of the information of the event access it and trigger the interaction (e.g., sharing the information of an event occurring) with other relevant agents. The different actions performed might create following events. With this consideration, we can first describe the different events considered in Domino and then map the events with the roles and agents which will be notified when that event occurs and with the roles/agents which generate the event.

We have done a conscious decision to minimise the coupling between the events (driven by the event-driven simulator) and the agents and their interactions. This ensures that the agent model itself can be re-used, extended or linked to different simulator engines in a simple manner.

We can see the information and the simulation driven by the different events that are linked to flight trajectory activities: submitting a flight plan, updating the flight plan and operating a trajectory (from gate-to-gate). In an event-driven simulation nothing is computed between events as the simulator advances from event to event. Therefore, as different actors need information on what is happening to the flight at different moments during its trajectory (e.g., the E-AMAN needs to be notified when a flight enters its planning horizon and when it enters its execution horizon), specific events to discretise the trajectory are needed. The number of events that are needed to discretise the trajectory might vary between different simulations depending of the needs of the different agents. Therefore, these events (Flight_Crossing_Point) have been defined in Domino as general and parametrisable as possible to allow this discretisation of the flight. These notifications of crossing points can either be at: top of climb (TOC), at top of descend (TOD), at a given distance from the origin (x nm from origin) or at a given distance to the destination (x nm to destination). This will allow us to sample the trajectory at different points, as needed by the agents. For this reason, we have created a Radar agent which distributes the submitted flight plan to interested agents (e.g., E-AMAN). These agents indicate when they want to be notified (e.g. x nm before the arrival at destination). When the flight reaches these milestones, the Radar agent will distribute the information to the interested parties (See diagram in Figure 15). This approach will allow to easily extend the capabilities of the model as if in the future other agents need to react to a flight reaching a given milestone in its trajectory, for example the airline operating centre being notified when the flight reaches the top of climb to reassess if changes in its trajectory are needed, they just need to subscribe to the required event that will be distributed by the Radar.

5.1.2 Events in Domino

With the previous consideration of generic crossing points in the trajectory in mind, the set of events identified in Domino are described in Table 46.

Table 46. Events in Domino simulation model

Event	When	Created by	Updated before it happens by	Read by	Triggers event
FP_submission	3 hours before EOBT a flight plan is submitted	System initialization	If EOBT is modified – Airline Operating Centre	Airline Operating Centre (Airline Flight Planner)	Pushback_Ready
Pushback_Ready	EOBT	Airline Operating Centre (Airline Flight Planner)	If EOBT is updated	Flight (Departing Slot Requester)	Pushback
Pushback	Pushback	Flight (Departing Slot Requester)	-	Flight (Aircraft Departing Handler) Airline Operating Centre (Passenger Reallocation)	Takeoff
Takeoff	Takeoff	Flight (Aircraft Departing Handler)	-	Flight (Operate Trajectory)	Flight_Crossing_Point (or Landed if no intermediate reporting point)
Flight_Crossing_Point	When flight reach a given point in its trajectory as defined by the agents that have interest on this information.	Flight (Operate Trajectory)	-	Radar (Disseminate Flight Position Update) Flight (Operate Trajectory)	Flight_Crossing_Point Landed
Landed	When flight reaches arrival runway	Flight (Operate Trajectory)	-	Flight (Ground Arrival Handler)	Flight_Arrival
Flight_Arrival	AIBT	Flight (Ground Arrival Handler)	-	Airline Operating Centre (Turnaround OperationS)	-

Figure 14 shows how events are related to each other for a given flight with a turnaround.

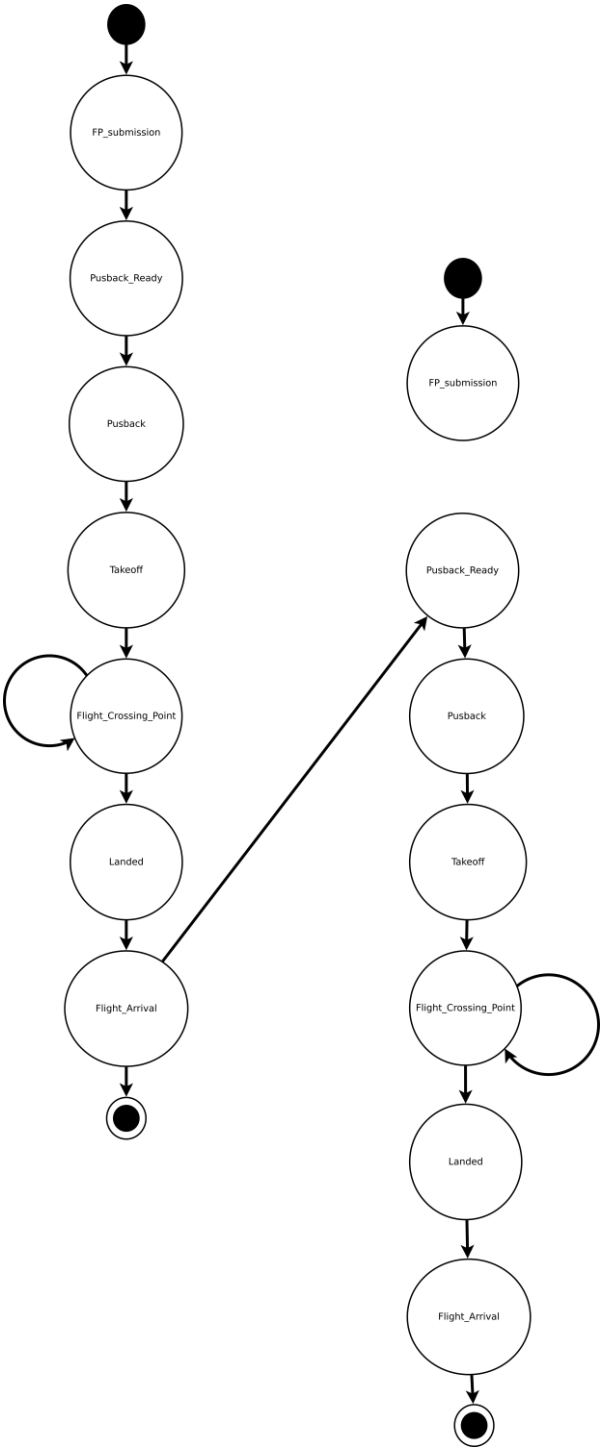

Figure 14: Events for a given flight with a turnaround

5.1.3 Sequence diagrams triggered by events

For each event that is considered in Domino, a sequence diagram is created showing the interactions that will create further events. Note that not all interactions are shown as exchanges with other agents are not displayed.

5.1.3.1 FP_submission

At flight plan submission, the AOC decides the flight plan for the flight (this might imply several interactions with the Network Manager, submitting a flight plan and being assigned an ATFM slot if needed). Once the Flight Plan is accepted, the Network Manager is notified, and it sends the Flight Plan to the Radar agent which distributes it to the parties interested (e.g., E-AMAN and DMAN, which update their arrival and departing queues). The agents interested in being notified when the flight reaches different stages report to the Radar at which points they want to be notified. The Radar creates the Flight_Crossing_Point events to meet these points of reporting and yield for them after sending them to the Network Manager, who in turn can send them to the AOC to be updated in the flight plan of the Flight. (i.e., the flight plan is enhanced with points where the flight needs to notify that it is there).

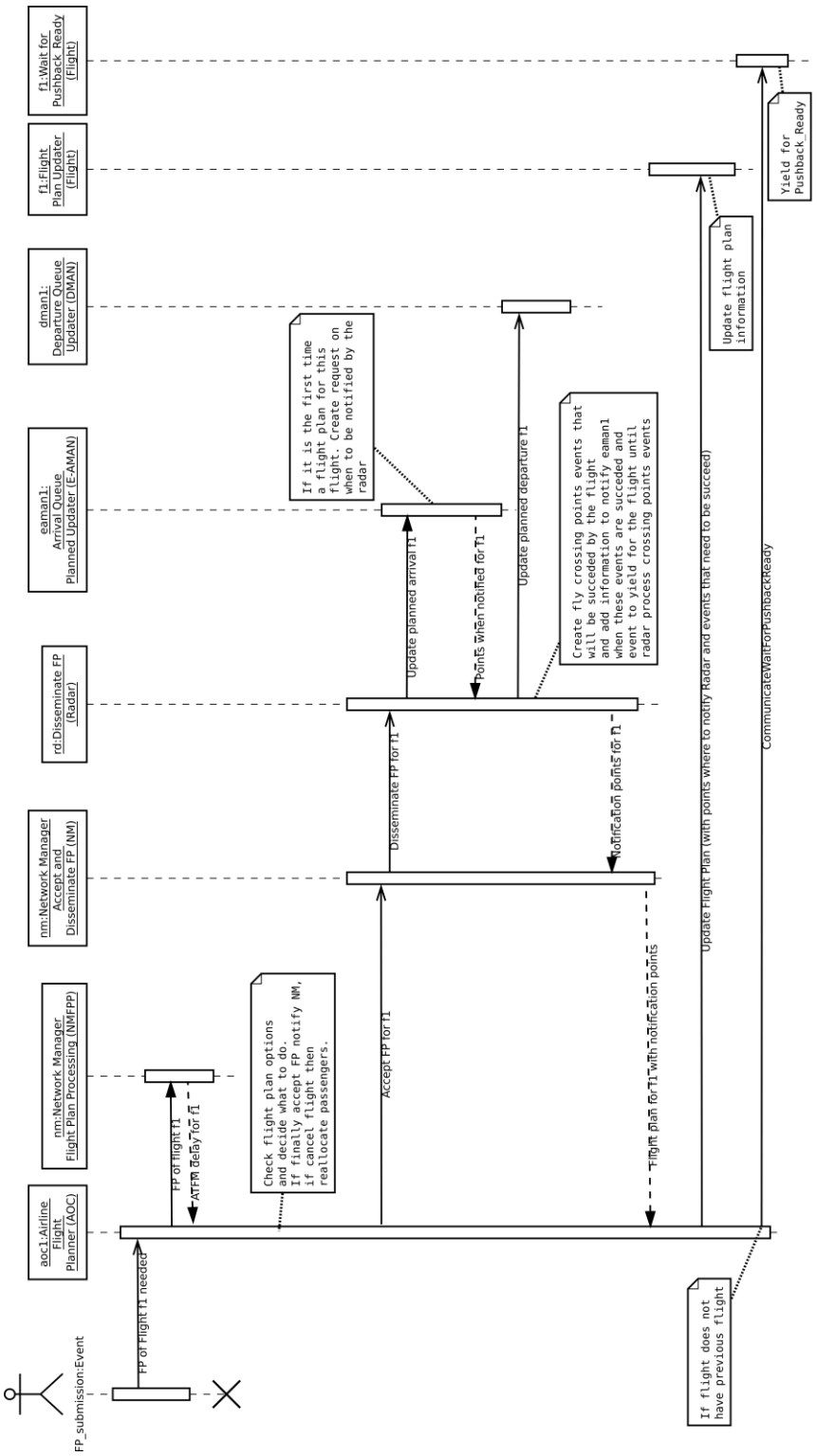

Figure 15: FP_submission event diagram

5.1.3.2 Pushback_Ready

At Pushback_Ready, the Flight requests a departing slot from the DMAN and a taxi out estimation to the Airport. With that information it computes the Pushback event.

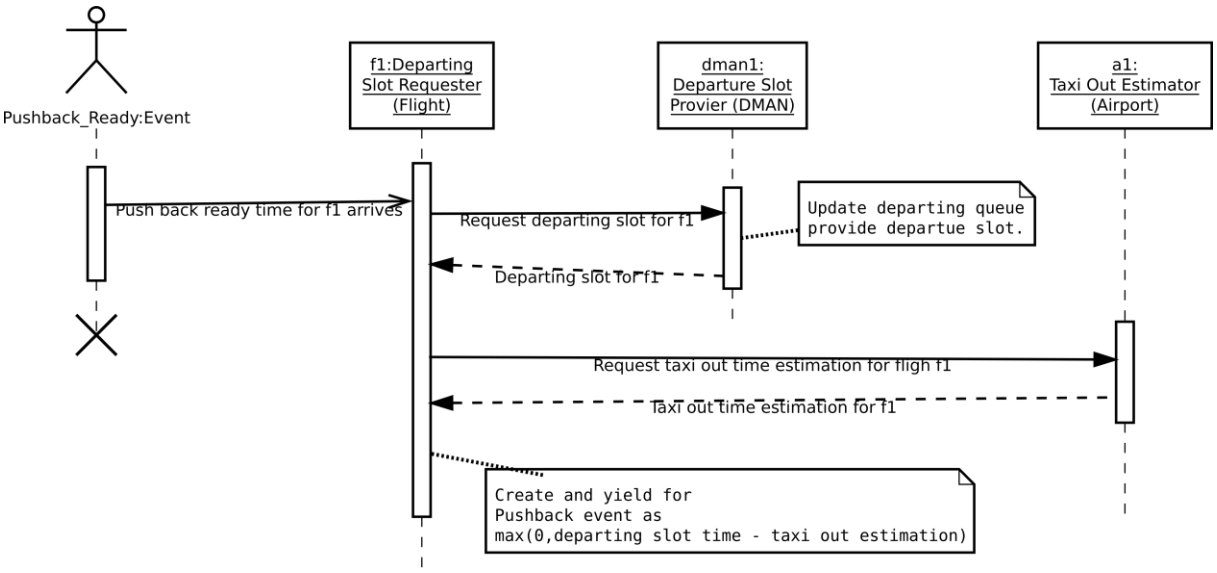

Figure 16: Pushback_Ready event diagram

5.1.3.3 Pushback

Two agents are interested in the Pushback event: The Flight in order to compute when it will do its take-off, and the Airline Operating Centre so to assess which passengers should have been in the flight and have missed their connection and rebook them.

The flight requests the taxi-out time and generates the Takeoff event.

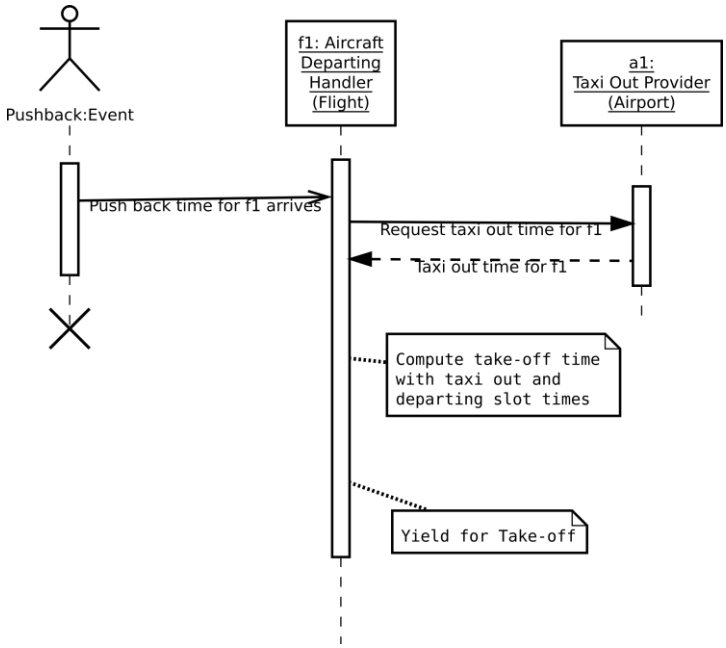

Figure 17: Pushback event diagram for Flight

The Airline Operating Centre checks which passengers have missed their connections and reallocates them by requesting the preferences to the Passenger agent.

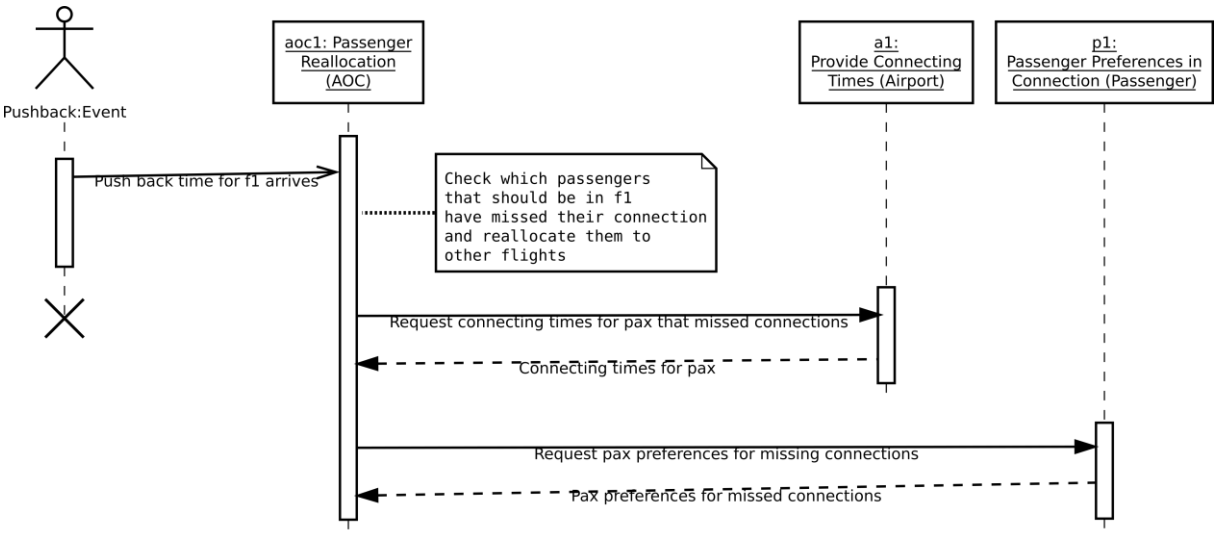

Figure 18: Pushback event diagram for Airline Operating Centre

5.1.3.4 Takeoff

At takeoff the flight starts the operating of the trajectory which will be executed in a loop until the flight lands. The Operate Trajectory computes the trajectory with the flight plan information available and the constraint (e.g., landing slot) for the trajectory. It checks which is the next reporting point and yields for it.

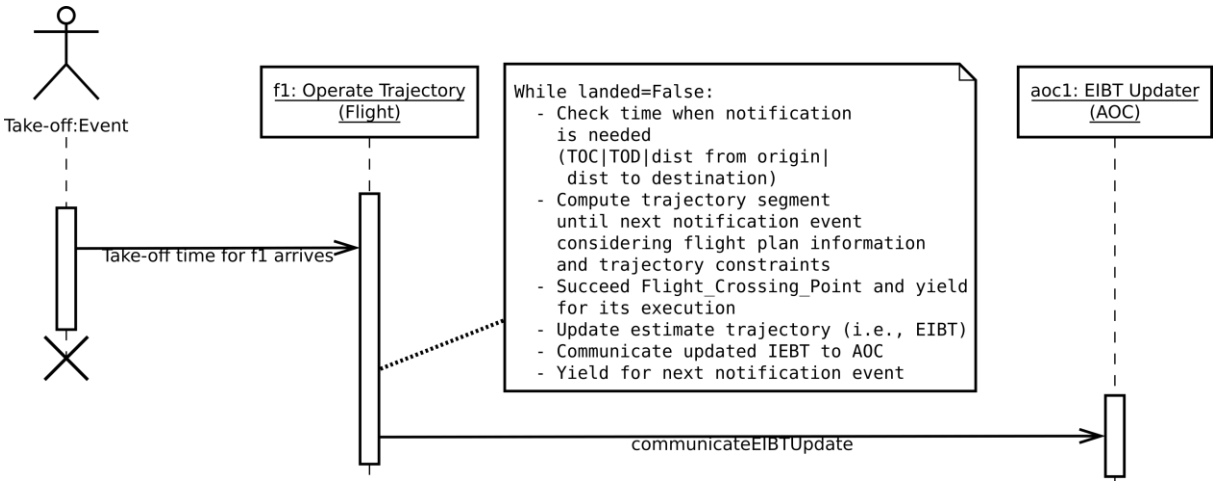

Figure 19: Takeoff event diagram

5.1.3.5 Flight_Crossing_Point

Every time the Operate Trajectory gets to a crossing point, it succeeds the Flight_Crossing_Point event which is captured by the Radar agent who disseminates the information to the relevant Agents. For example, in the diagram below, it updates the E-AMAN that the flight has entered its planning horizon. The E-AMAN will then update the arrival slot of the flight, who will use this information to compute the next iteration of Operate Trajectory.

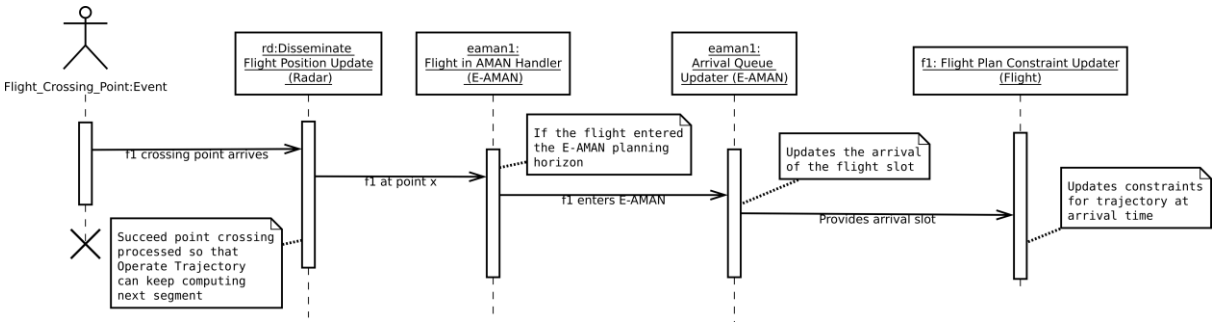

Figure 20: Flight_Crossing_Point event diagram

5.1.3.6 Landed

When a flight lands, the Flight requests the Airport the taxi-in time and computes the AIBT. It then creates the Flight_Arrival event.

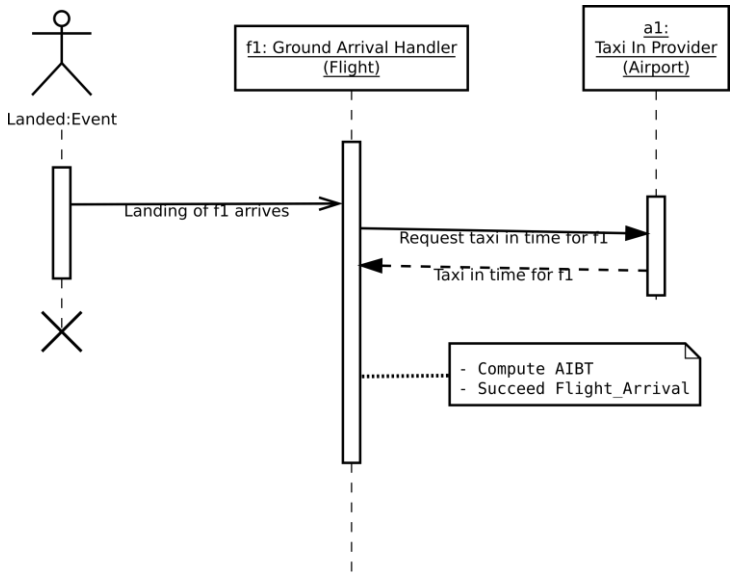

Figure 21: Landed event diagram

5.1.3.7 Flight_Arrival

The Airline Operating Centre is notified by the Flight_Arrival event and performs the turnaround operations. This includes processing the passengers which have arrived: if connecting passengers have missed their connection, rebook them. Manage with the Airport the ground handling processes to compute the turnaround time. Update the information of the EOBT of the next rotation of the flight and update the flight plan of the next rotation which flight will yield for the Pushback_Ready event.

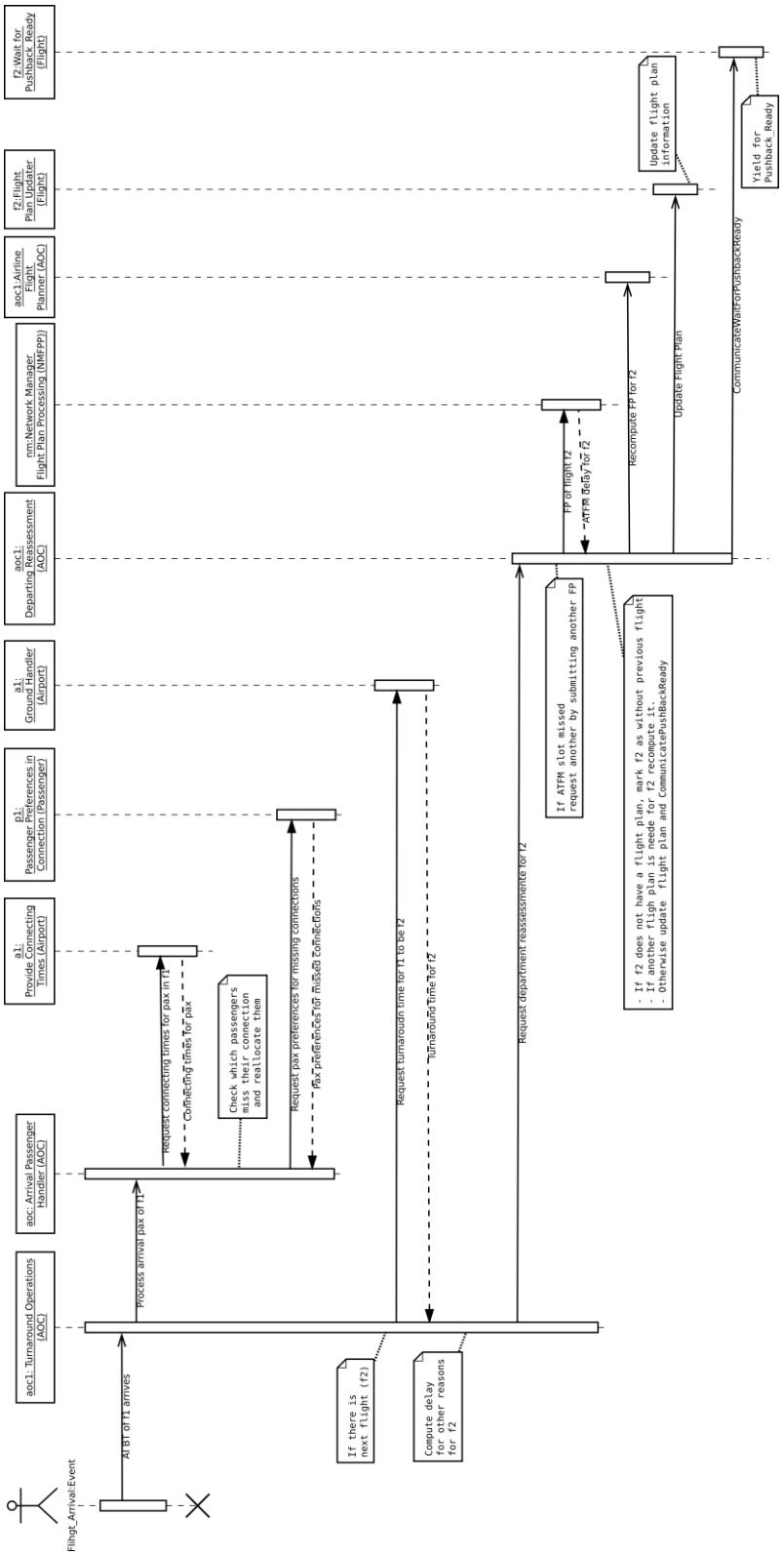

Figure 22: Flight_Arrival event diagram

5.2 Communication channels

Since the essence of an agent-based model is to create routines with private memory whose accesses are restricted, one needs to create communication channels between the agents. Communication is opposed to direct access to a resource. For instance, if the network managers needs the flight plan of a given a flight, it should not access it directly in the airline memory. It should instead ask for the flight plan (which is a communication) and wait for the flight plan sent by the airline (another communication).

In practice, there are several ways of implementation communication channels. One approach is to use dedicated communication protocols such as FIPA [4] which define the fields of the messages and an ontology which describes the elements. This approach can be more generic but require extra computation. Communications can be performed over a server allowing the distribution of agents across platforms or emulated by calling dedicated functions in the agent objects.

In Python, few ABM frameworks have been developed, and even fewer include dedicated communication procedures. An exception is SPADE, a python module which uses the XMPP standard (used for instant chat services for example) to allow easy communication through a central server [10]. This module has been tested and it is unfortunately too slow for than what would be requested for Domino. According to our test, a single message takes between 0.002 and 0.02 second to be processed. Domino will likely feature around 100k agents exchanging 3M messages in a single simulations, which would take up to 16 hours only to send messages. As a comparison, a direct memory access is below 10-5 s, i.e., more than 1000 times faster.

Because we are interested in developing a model which can be easily customised and modified in the future, we maintain the principle of communication channels as they help to standardise the interaction between agents and help decoupling the different components of the system from an implementation point of view. As a consequence, a simple communication process in Python will be developed. This will allow any agent to have an 'address' and a 'letterbox' to which other agents can send messages. A dedicated 'postman' agent checks letterboxes for messages and deliver them to the right address, essentially playing the role of a server. This framework allows to modify the delivery process without changing the agents' behaviours. This could be useful for instance if in the future one wants to use to the model with human agents which acts on the system independently. More generally, it allows optimisation opportunities. Finally, the ad-hoc postman agent could be replaced by a server (such as in the SPADE framework) seamlessly if desired in the future.

5.3 Sequential, concurrent and parallel programming

One of the advantages of an event-driven simulation is that it naturally leads to use concurrent programming. Compared to sequential programming, concurrent programming allows several routines (called co-routines) to run independently from each other, and not one after the other. This

allows to program processes which conceptually run in parallel. For instance, while a flight is flying, another one prepares for push-back: these two processes are independent and can conceptually happen at the same time.

Concurrent programming is distinct from parallel programming. In the former, tasks are conceptually parallel but are executed sequentially slice by slice by the processor. In particular, it implies that there is no gain of real time by running the code sequentially or concurrently: everything happens in the same process. In parallel programming, routines are physically running in different processes, which allows a reduction of the running time if several processors are available. Parallel computing is very powerful, but requires synchronisation between processes, to be sure that everything that is needed to run a part of the code has been computed previously. This synchronisation is not trivial to perform.

Domino's model will be concurrent but not parallel. The reason is simply that the project would not gain anything from a parallelisation. Indeed, Domino's model, by nature, is to be run several times to have reliable results, since it is stochastic. Moreover, launching the same simulation on several cores is absolutely trivial. These different executions will run in different processes that would not wait for each other, thus utilising fully the computational power of the several cores. On the contrary, intra-model parallelisation requires a lot of effort and processes would routinely wait for each other, leading to smaller gains in computing power.

Note however that a parallelisation could be quite easily achieved in the future with the architecture chosen for the model. This can be useful in the future to use the model as real-time simulator where humans are in the loop. In particular, the dedicated communication channel developed by the team can be easily changed a more powerful one, for instance SPADE.

6 Implementation details

Domino follows the principles of clean code, believing it to be prevalent in its computational power. This will allow more flexibility and reusability in the future. As a consequence, Domino chose the widely-used Python programming language for the development of the model. Python is an open-source language which has slowly become a standard in the industry and research. It is stable and reliable and comes with an extensive list of free modules to be used and enhanced by the community. These include high-performing libraries for numeric calculations based on Fortran and C routines. The aforementioned characteristics of Python, as well as its object-oriented features and high readability, make it particularly well suited for agent-based modelling.

Different standard modules will be used for the model, and more advanced ones are also considered. Of note is SimPy, which allows for easy managing of event-driven simulations with concurrent processes and shared resources. Another potential module of interest if MESA, built to easily create and simulate various types of ABMs. As mentioned in section 5.2., other modules, such as SPADE, have been tested, but have been discarded due to performance issues.

The code itself will be newly developed. However, parts of the code will be directly imported from other models, in particular Mercury. Mercury is an even-driven simulator developed over the past 9 years which has been rewritten in Python during the recent Vista project. Domino model can be viewed as an agentification of Mercury, and thus might heavily reuse parts of its code.

Running the implemented model will be done mostly on standard laptops and desktop computers during the development phase. However, during the production phase (the final simulation runs), Domino may make use of more powerful servers. That might be done:

- either using some commercial cloud computing solution like Amazon Web Services (AWS),
- or using the University of Westminster's own cluster.

Different tests will be performed during development to decide which solution, if any, is better for the project.

A development environment for Domino will be created and shared among the different partners so that compatible Python modules are used ensuring that the code can be run at the different facilities and easily developed on different machines. The code is shared among the different developers using a GitHub repository.

Finally, Domino project is committed to producing highly reliable scientific results. In order to do this, the partners have developed over the years several ways to getting consistent and reliable inputs, as well as traceable outputs. This is mainly done by using a MySQL database, hosted at UoW. This database has been described in more details in D2.1. It will used by the model to pull out the input and drop raw results. Analysing the results will then be done by querying the database.

7 Next steps and look ahead

This deliverable described in detail how the model will be built. The next immediate step is the implementation of the model itself. A first version will be implemented, on which the different scenarios will be run. In parallel, new metrics considering network effects, e.g., causality and centrality metrics, will be developed and presented in deliverable D5.1 – Metrics and analysis approach. The metrics will then be used on the output of the model and presented at a dedicated workshop: stakeholders investigative case studies and adaptive case studies definition workshop. The model results, as well as the feedback from the workshop, will be presented in two deliverables (D5.2 – Investigative case studies results and D6.3 – Workshop results summary). After that, the adaptive case studies will be run, where the system is optimised using the model and the metrics previously developed.

8 References

1. Domino Consortium, 2018. D3.1 Architecture definition – Ed.01.00.00
2. Domino Consortium, 2018. D3.2 Investigative case studies description
3. SJU, 2018. EATM Portal - https://www.eatmportal.eu
4. Foundation for Intelligent Physical Agents, 2012. FIPA ACL Message Structure Specification
5. Grimm V., Berger U., DeAngelis D.L., Gary Polhill J.G., Giske J. and Railsback S. F., 2010, The ODD protocol: A review and first update, Ecological Modelling 221, 2760-2768
6. Grimm V., Berger, U., Bastiansen F., Eliassen S., Ginot V., Giske J., Goss-Custard J., Grand T., Heinz S.K., Huse, G., Huth A., Jepsen J.U., Jørgensen C., Mooij W.M., Müller B., Pe'er G., Piou C., Railsback S.F., Robbins A.M., Robbins M.M., Rossmanith E., Rügger N., Strand E., Souissi S., Stillman R.A., Vabø R., Visser U. and DeAngelis D.L., 2006, A standard protocol for describing individual-based and agent-based models, Ecological Modelling 189, 115-126
7. Luke S., 2015. Multiagent Simulation and the MASON Library - Manual Version 19. Department of Computer Science George Mason University
8. Montagna S., Omicini A. and Pianini D., 2015. Extending the Gillespie's Stochastic Simulation Algorithm for Integrating Discrete-Event and Multi-Agent Based Simulation. In proceedings of the Multi-Agent-Based Simulation XVI - International Workshop, MABS 2015, Istanbul, Turkey
9. Siebers P.O., Macal C.M., Garnett J., Buxton D. and Pidd M., 2010. Discrete-Event Simulation is Dead, Long Live Agent-Based Simulation!'. Journal of Simulation, 4(3) pp. 204-210.
10. SPADE, 2018. Smart Python Agent Development Environment. https://spade-mas.readthedocs.io/en/latest/readme.html
11. Wooldrige M., Jennings N.R. and Kinny D., 2000, The Gaia methodology for agent-oriented analysis and design, Autonomous Agents and Multi-Agent Systems, 3, 285-312

9 Acronyms

4DTA: 4D Trajectory Adjustment

ATM: Air traffic management

E-AMAN: Extended Arrival Manager

FAC: Flight Arrival Coordination

FP: Flight prioritisation

H2020: Horizon 2020 research programme

SESAR: Single European Sky ATM Research

SJU: SESAR Joint Undertaking

TBO: Trajectory-Based Operations

UDPP: User-Driven Prioritisation Process

1 Annex – Roles model

Table 47. Role schema description Airline Flight Planner (AFP)

Role schema	Airline Flight Planner (AFP)
Description	Select which flight plan will be executed. This means selecting the 4D trajectory (including delayed departing time (e.g., waiting for passengers)) and possibly cancelling flight (i.e., the Airline Flight Planner decides to cancel the flight plan). 5. Check which flights are ready for pre-departure (less than 3 hours before their EOBT and in scheduled status) or get a request to recompute the flight plan selection of a given flight. 6. Submit flight plan which might lead to ATFM delay update. 7. Check different flight plan options and their costs. 8. Decide which flight plan to execute tactically. The liveness of AFP will depend on the mechanism 4DT and the FP implementation. The action of RequestATFMSlot will depend on the FP mechanism: • FP_0: Request ATFM slot • FP_1 and FP_2: Request ATFM slot and then prioritise the flight and optionally request a flight swap. The action of DecideOption depends on 4DT: • 4DT_0: Rule of thumb to decide between speed-up, wait-for-passengers and cancellation • 4DT_1 and 4DT_2: Selection of option based on cost provided by CheckFPOption
Protocol and activities	FlightsReadyForPreDeparture, *WaitForRequestFlightPlan*, *RequesATFMSlot*, *WaitForATFMSlot*, *RequestFPCostOptionsEstimation*, *WaitForFPCostOptions*, DecideOption, *RequestRelocatePax*, RecomputeEOBT, RecomputeEIBT, CheckAlreadyATFM, *RequestFlightPrioritistaion*, *WaitForFlightPriorities*, *RequestFlightSwap*, *WaitForFlightSwap*, *CommunicateFPtoFlight*, ComputePushBack, *CommunicateWaitForPushbackReady*, *AcceptFP, CancelFP, WaitForFPAcceptance*

Permissions	reads *flight ready for pre-departure // flights which are 3 hours before their EOBT with status on scheduled* *flight details // all details of the flight to be computed the cost* *flight status // status of the flight (scheduled, pre-departure)* *flight ATFM delay // ATFM delay of the flight* *flight credits // for FP_2* generates *flight plan to be executed tactically* updates *ATFM delay of flight* *EOBT of flight* *COBT of flight* *EIBT of flight* *flight status to pre-departure*
Liveness responsibilities	AFP = ((FlightsReadyForPreDeparture | *WaitForRequestFlightPlan*).UpdateFP+.(*CancelFP* || *RequestRelocatePax*) | (*RecomputeEIBT.AcceptFP.WaitForFPAcceptance*).*CommicateFPtoFlight*.[*CommunicateWaitForPushbackReady*])^{w} UpdateFP for 4DT_0 = *RequestATFM*.DecideOption UpdateFP for 4DT_1 and 4DT_2 = *RequestATFM.RequestFPCostOptionsEstimation.WaitForFPCostOptions*.DecideOption RequestATFM for FP_0 = *CheckAlreadyATFM* | *RequestATFMSlot.WaitForATFMSlot* RequestATFM for FP_1 and FP_2 = CheckAlreadyATFM | *RequestATFMSlot.WaitForATFMSlot.RequestFlightPrioritisation*.WaitForFlightPriorities.[*RequestFlightSwap.WaitForFlightSwap*] DecideOption for 4DT_0 = Apply rules of thumb to decide trajectory and wait-for-passengers. DecideOption for 4DT_1 and 4DT_2 = Select option with expected lower cost
Safety responsibilities	• DecideOption = change flight plan → (UpdateFP) redone • DecideOption = cancel flight → RequestRelocatePax executed • If ATFM delay is given then COBT = EOBT • CommunicateWaitForPushbackReady if flight does not have previous flight or already had push back.

Table 48. Role schema description Network Manager Flight Plan Processing (NMFPP)

Role schema	Network Manager Flight Plan Processing (NMFPP)
Description	Process a flight plan submission by the NM. It checks if ATFM delay is needed and if it is the case returns the ATFM delay.
Protocol and activities	*WaitForFPProcessingRequest,* ComputeATFMDelay, *ReturnATFMDelay*
Permissions	reads *information on airspace conditions // potential delay and congestion* *flight demand // demand of flights in the airspace* generates *ATFM flight delay*
Liveness responsibilities	NMFPP = (*WaitForFPProcessingRequest*.ComputeATFMDelay. *ReturnATFMDelay*)w
Safety responsibilities	Ensure capacities are maintained pre-tactically

Table 49. Role schema description Network Manager Accept and Disseminate FP (NMAD)

Role schema	Network Manager Accept and Disseminate FP (NMAD)
Description	Request the dissemination of the Flight Plan to the entities interested in it and returns the points where the Flight needs to notify when reaching them.
Protocol and activities	*WaitForAcceptanceOfFP, RequestDisseminationOfFP, WaitForDisseminationOfFP, ReturnPointsNotificationFP*
Permissions	true
Liveness responsibilities	NMAD = (*WaitForAcceptanceOfFP.RequestDisseminationOfFP.WaitForDisseminationOfFP. ReturnPointsNotificationFP*)^{w}
Safety responsibilities	true

Table 50. Role schema description Network Manager Cancel FP (NMC)

Role schema	Network Manger Cancel FP (NMC)
Description	Request the cancellation of a Flight Plan. Request the dissemination of the cancellation of a flight plan.
Protocol and activities	*WaitForCancellationOfFP, RequestDisseminationOfCancellationFP*
Permissions	true
Liveness responsibilities	NMC = (*WaitForCancellationOfFP.RequestDisseminationOfCancellationFP*)^{w}

Safety responsibilities	true

Table 51. Role schema description Disseminate FP (DFP)

Role schema	Disseminate FP (DFP)
Description	Send the FP to the entities interested in it to get the points when it need to be notified (E-AMAN and DMAN)
Protocol and activities	*WaitForFPToDisseminate, UpdatePlannedArrivalFP, UpdatePlannedDepartureFP, WaitForPointsNotificationArrival, ReturnPointsNotificationFP*
Permissions	reads *information on flight plan to know origin (DMAN) and destination (E-AMAN)* generate *list of interested entities in Flight Plan*
Liveness responsibilities	DFP = (*WaitForFPToDisseminate.UpdatePlannedArrivalFP. WaitForPointsNotificationArrival.UpdatePlannedDepartureFP. ReturnPointsNotificationFP*)^{w}
Safety responsibilities	Link notification of crossing points with Flight Plan and DMAN/E-AMAN

Table 52. Role schema description Disseminate Cancellation FP (DCFP)

Role schema	Disseminate Cancellation FP (DCFP)
Description	Send the information that a FP has been cancelled to the entities interested in so that its slots are released (E-AMAN and DMAN)
Protocol and activities	*WaitForFPToDisseminateCancellation, CancellationArrivalFP, CancellationDepartureFP*
Permissions	reads *information on flight plan to know origin (DMAN) and destination (E-AMAN)* update *list of interested entities in Flight Plan*
Liveness responsibilities	DCFP = (*WaitForFPToDisseminateCancellation.CancellationArrivalFP. CancellationDepartureFP*)^{w}
Safety responsibilites	Removes from list of interested entities in Flight Plan the information related to the Flight Plan

Table 53. Role schema description Departure Queue Updater (DQU)

Role schema	Departure Queue Updater (DQU)
Description	When a flight updates its EOBT, its position in the departure queue is updated.
Protocol and activities	*WaitForDepartureQueueUpdateRequest*, UpdateDepartureQueue
Permissions	reads *departure queue // queue with flights waiting to depart* *taxi out times* updates *departure queue*
Liveness responsibilities	DQU = (*WaitForDepartureQueueUpdateRequest*.UpdateDepartureQueue)^{w}
Safety responsibilities	• Capacity of departure runway is maintained • Slot in queue >= EOBT + taxi out • Slots already assigned are not changed • Flight marked as planned in the queue • If update is cancellation remove flight from list of departing queue

Table 54. Role schema description Arrival Queue Planned Updater (AQPU)

Role schema	Arrival Queue Planned Updater (AQPU)
Description	Update the queue of flights planned to arrive with information from the AOC when a flight update its EIBT. If it is the first time the flight is provided then send the requests of points where the E-AMAN needs to be notified of the flight.
Protocol and activities	*WaitForArrivalQueueUpdateRequest*, UpdateArrivalQueue, *ReturnRequestPointsNotification*
Permissions	reads *arrival queue // with slots in the arrival queue* *taxi in time estimation* *execution horizon distance* *planning horizon distance* updates *arrival queue by updating the flight in the queue*
Liveness responsibilities	AQPU = (*WaitForArrivalQueueUpdateRequest*.UpdateArrivalQueue.[*ReturnRequestPointsNotification*])^{w}

Safety responsibilities	• Capacity of arrival runway is maintained • Slot in queue >= EIBT - estimated taxi in • Slots assigned are not changed • Update only *scheduled* flights • If it is the first time the flight is introduced in the list then return request of points where to be notified at destination - planning horizon and destination - execution horizon. • If update is cancellation remove flight from list of departing queue

Table 55. Role schema description Passenger Reallocation (PR)

Role schema	Passenger Reallocation (PR)
Description	Decide how to manage connecting passengers when a flight has left. 3. Check passenger that should have been in the flight that has left and have missed their connection 4. Rebook them onto following flights to destination, compensate and return them to destination, pay for care, and potentially put them in hotels by checking preferences with passengers.
Protocol and activities	*WaitForRelocatePassengers,* CheckFlightPushBack*,* ComputeMissedConnectingPassengers*, RequestPaxPreferenceRebooking, WaitForPaxPreferencesRebooking,* RelocatePassengers*,* CompensatePassengers*, RequestMinimumConnectingTimes, WaitForMinimumConnectingTimes,* ComputeReallocationOptions
Permissions	reads *passenger itineraries // to know connections* *flights and their status // status of flights to know where they are (scheduled, pre-departure, cancelled)* updates *passenger itineraries*
Liveness responsibilities	PR = ([*WaitForRelocatePassengers*|CheckFlightPushBack). ComputeMissedConnectingPassengers. (*RequestMinimumConnectingTimes*.*WaitForMinimumConnectingTimes*)+. ComputeReallocationOptions.*RequestPaxPreferenceRebooking*. *WaitForPaxPreferencesRebooking*.RelocatePassengers.CompensatePassengers)^{w}
Safety responsibilities	• Passengers who missed their connection relocated to flights with status 'scheduled' or 'pre-departure' • All flights load factor < 100% • All passenger itineraries allocated to flights and (possibly) booked into hotel and compensated / duty of care applied

Table 56. Role schema description Passenger Preferences in Connection (PPC)

Role schema	Passenger Preferences in Connection (PPC)
Description	Provides the preferences of passengers for follow up connections when missed connection.
Protocol and activities	*WaitPassengerPreferencesConnectionRequest, ReturnPassengerPreferencesConnection,* ComputePassengerPreferencesinConnection
Permissions	reads *options available for passenger with missed connections* generates *preferences of passengers for connections*
Liveness responsibilities	PPC = (*WaitPassengerPreferencesConnectionRequest*. ComputePassengerPreferencesinConnection. *ReturnPassengerPreferencesConnection*)^{w}
Safety responsibilities	Preferences valid, capacities maintained.

Table 57. Role schema description Provide Connecting Times (PCT)

Role schema	Provide Connecting Times (PCT)
Description	Provides connecting times for passengers at airport
Protocol and activities	*WaitForConnectingTimeRequest, ReturnConnectingTimes,* EstimateConnectingTimes
Permissions	reads *MCT for passengers at airport* *passenger types* generates *connecting times for passengers*
Liveness responsibilities	PCT = (*WaitForConnectingTimeRequest*.EstimateConnectingTimes. *ReturnConnectingTimes*)^{w}
Safety responsibilities	• Returns minimum connecting time (MCT) and connecting time estimated (CT) • CT>=MCT

Table 58. Role schema description Airline Pre-departure Flight Prioritiser (ADFP)

Role schema	Airline Pre-departure Flight Prioritiser (ADFP)
Description	Compute the priority of a flight. Used in FP_1 and FP_2. ComputeFlightPriority in FP_2 considers also the use of the credit system.

Protocol and activities	*WaitFlightPrioritisationRequest*, ComputeFlightPriority, *ReturnFlightPriority*
Permissions	reads *supplied flight to prioritise // identifies of flight to which the priority will be computed* *flight details // all details of the flight to be computed the priority* *all flights expected delay // expected delay for all flights in the airline* *all flights load factors // load factors of the flights in the airline* *passenger itineraries // passengers itineraries and connections* *minimum connecting times // time required to do connections of passengers* *aircraft performances // aircraft performances of the flight* *information on airspace conditions // potential delay and congestion, routes options* generates *the priority of the flight*
Liveness responsibilities	ADFP = (*WaitFlightPrioritisationRequest*.ComputeFlightPriority.*ReturnFlightPriority*)^{w}
Safety responsibilities	true

Table 59. Role schema description Flight Swapper (FS)

Role schema	Flight Swapper (FS)
Description	Swap flights with ATFM delay for flight prioritisation mechanisms. Behaviour dependent on level of FP mechanism: • FP_1: Use of priorities provided by flights. Allow swapping of flights within an airline. • FP_2: Use of priorities provided by flights. Allow swapping of flights between airlines.
Protocol and activities	*WaitForFlightSwapRequest*, ComputeFlightSwap, *ReturnFlightSwap*
Permissions	reads *flight prioritisation // prioritisation of flight* *flight credits // for FP_2* generates *flight swap*
Liveness responsibilities	FS = (*WaitForFlightSwapRequest*.ComputeFlightSwap.*ReturnFlightSwap*)^{w}
Safety responsibilities	Allow swapping if valid

Table 60. Role schema description Airline Pre-departure Flight Plan Cost Computer (AFPC)

Role schema	Airline Pre-departure Flight Plan Cost Computer (AFPC)
Description	Compute cost of changing the 4D trajectory of a given flight at pre-departure stage. Provides the best option and cost for each of the change possibilities: wait for passengers, modifying 4D trajectory, cancelling flight. The role is different depending on 4DT: • 4DT_1 and 4DT_2: Same liveness but different degree of depth of analysis, since 4DT_2 considers downstream effects.
Protocol and activities	*WaitFPCostEstimationRequest,* ComputationCostWaitForPax, ComputaionCostChangeTrajectory, ComputationCostCancellation, *ReturnFPCostPerOption*
Permissions	reads *supplied flight to compute cost // identifies of flight to which compute the different options* *flight details // all details of the flight to be computed the cost* *all flights expected delay // expected delay for all flights in the airline* *all flights load factors // load factors of the flights in the airline* *passenger itineraries // passengers itineraries and connections* *minimum connecting times // time required to do connections of passengers* *aircraft performances // aircraft performances of the flight* *information on airspace conditions // potential delay and congestion, routes options* generates *for each option the new flight plan and their cost*
Liveness responsibilities	AFPC = (*WaitFPCostEstimationRequest*.EstimateCostOptions.*ReturnFPCostPerOption*)^{w} EstimateCostOptions = (ComputationCostWaitForPax || ComputaionCostChangeTrajectory || ComputationCostCancellation)
Safety responsibilities	true

Table 61. Role schema description Flight Plan Updater (FPU)

Role schema	Flight Plan Updater (FPU)

Description	When the flight plan is changed the information is updated for the flight. 4. Wait for request of update flight plan 5. Update information on flight plan including when to report 6. If the flight is airborne then request a flight parameters recomputation
Protocol and activities	*WaitForFPUpdate*, UpdateFPInformation
Permissions	updates *flight plan // information on flight plan - speed, distances, being cancelled, etc.*
Liveness responsibilities	FPU = (*WaitForFPUpdate*.UpdateFPInformation)^{w}
Safety responsibilities	• If flight status = landed then do not change the FP • Flight cannot be cancelled if flight status <> pre-departure

Table 62. Role schema description Wait for Pushback_Ready (WPBR)

Role schema	Wait for Pushback_Ready (WPBR)
Description	When the flight is ready for departure then wait for push back ready event
Protocol and activities	*WaitForWaitForPushbackReadyRequest,* WaitForPushbachReady
Permissions	reads *pushback_ready time*
Liveness responsibilities	WFPBR= (*WaitForWaitForPushbackReadyRequest*.WaitForPushbachReady)^{w}
Safety responsibilities	Pushback_Ready time computed

Table 63. Role schema description Departure Slot Requester (DSR)

Role schema	Departure Slot Requester (DSR)
Description	When the time of the EOBT ready aircraft arrives then the departing slot is requested. 3. Request departure slot 4. Request estimate taxi-out time
Protocol and activities	CheckFlightPushBackReady, *RequestDepartingSlot, WaitForDepartingSlot, RequestTaxiOutEstimation, WaitForTaxiOutEstimation* , ComputePushBackTime

Permissions	reads *flight status // (ready)* *flight departing slot // departing slot of flight* *flight plan // information on flight plan - speed, distances, etc.* *updates flight status to pushback ready* *flight AOBT* generates *departing slot for the flight* *TakeoffTime flight*
Liveness responsibilities	DSR = (CheckFlightPushBackReady.(*RequestDepartingSlot.WaitForDepartingSlot* || *RequestTaxiOutEstimation.WaitForTaxiOutEstimation*).ComputePushBackTime)^{w}
Safety responsibilities	• AOBT = Final EOBT = Pushback time • AOBT = max(EOBT, Departing slot - taxi out estimation)

Table 64. Role schema description Taxi Out Estimator (TOE)

Role schema	Taxi Out Estimator (TOE)
Description	Provides an estimation of the taxi out time required for a flight
Protocol and activities	*WaitForTaxiOutEstimationRequest, ReturnTaxiOutEstimation,* EstimateTaxiOutTime
Permissions	reads *taxi out times distributions* *flight characteristics* generates *taxi out estimation*
Liveness responsibilities	TOE = (*WaitForTaxiOutEstimationRequest*.EstiamateTaxiOutTime. *ReturnTaxiOutEstimation*)^{w}
Safety responsibilities	true

Table 65. Role schema description Departure Slot Provider (DSP)

Role schema	Departure Slot Provider (DSP)
Description	When a flight is ready at the gate, sends a WaitForDepartureRequest to get the slot in the runway sequence. The role gets the request with the CTOT (if any) and provides a departure slot in the runway sequence. The role takes into account the taxi-out times and the order of the flights in the departure queue.

Protocol and activities	*WaitForDepartureRequest*, UpdateDepartureQueue, *ReturnDepartureTime*
Permissions	reads *departure queue // queue with flights waiting to depart* updates *departure queue*
Liveness responsibilities	DSP = (*WaitForDepartureRequest*.UpdateDepartureQueue. *ReturnDepartureTime*)^{w}
Safety responsibilities	• Capacity of departure runway is maintained • When a flight request a departure slot it is given • Departure as close as possible as CTOT • Slot provided >= Pushback Ready + taxi out

Table 66. Role schema description Aircraft Departing Handler (ADH)

Role schema	Aircraft Departing Handler (ADH)
Description	When the time of the push back arrives then the taxi out and the take-off time are computed. 3. Request taxi-out time 4. Compute takeoff time
Protocol and activities	*CheckFlightPushBack*, ComputeTakeoffTime, *RequestTaxiOutTime, WaitForTaxiOutTime*
Permissions	reads *Pushback event* updates *flight status to departed* *flight AOBT* generates *taxi out time*
Liveness responsibilities	ADH = (CheckFlightPushBack.*RequestTaxiOutTime*.*WaitForTaxiOutTime*. ComputeTakeoffTime)^{w}
Safety responsibilities	AOBT = Pushback time

Table 67. Role schema description Taxi Out Provider (TOP)

Role schema	Taxi Out Provider (TOP)
Description	Provides the actual taxi out time for a flight
Protocol and activities	ComputeTaxiOutTime, *WaitForTaxiOutRequest, ReturnTaxiOutTime*

Permissions	reads *aircraft type* generates *taxi out time*
Liveness responsibilities	TOP = (*WaitForTaxiOutRequest*.ComputeTaxiOutTime.*ReturnTaxiOutTime*)^{w}
Safety responsibilities	true

Table 68. Role schema description Operate Trajectory (OT)

Role schema	Operate Trajectory (OT)
Description	Fly until the next notification Flight Cross Point or Landing.
Protocol and activities	CheckFlightTakeOff, CheckNextFlightCrossPoint, ComputeTrajectorySegment, *WaitNextFlightReported*, NotifyFlightCrossPoint, NotifyLanded, *CommunicateUpdateEIBT*
Permissions	reads *flight trajectory* *flight constraints* *flight cross points* *aircraft performancesCommunicateUpdateEIBT* *flight plan information*
Liveness responsibilities	OT = ([CheckNextFlightCrossPoint | CheckFlightTakeOff]. WaitNextFlightReported.ComputeTrajectorySegment. *CommunicateUpdateEIBT*.[NotifyFlightCrossPoint.NotifyLanded])^{w}
Safety responsibilities	Last notification is Landed

Table 69. Role schema description EIBT Updater (EU)

Role schema	EIBT Updater (EU)
Description	Update the EIBT of a flight for the AO
Protocol and activities	*WaitForEIBTUpdate*,UpdateEIBTofFlight
Permissions	updates *flight information*
Liveness responsibilities	EU = (*WaitForEIBTUpdate*.UpdateEIBTofFlight)^{w}
Safety responsibilities	true

Table 70. Role schema description Strategic Departure Queue Builder (SDQB)

Role schema	Strategic Departure Queue Builder (SDQB)

Description	Builds the departure queue at an airport based on the flight schedules and airport departure capacity
Protocol and activities	CreateDepartureQueue
Permissions	reads *airport departure capacity* generates *departure queue*
Liveness responsibilities	SDQB = CreateDepartureQueue
	Capacity of departure runway is maintained

Table 71. Role schema description Strategic Arrival Queue Builder (SAQB)

Role schema	Strategic Arrival Queue Builder (SAQB)
Description	Builds the arrival queue at an airport based on the flight schedules and airport arrival capacity
Protocol and activities	CreateArrivalQueue
Permissions	reads *airport arrival capacity* generates *arrival queue*
Liveness responsibilities	SAQB = CreateArrivalQueue
Safety responsibilities	Capacity of arrival runway is maintained

Table 72. Role schema description Ground Arrival Handler (GAH)

Role schema	Ground Arrival Handler (GAH)
Description	When a flight has landed, this role computes when it will arrive at the gate. 3. Compute the taxi-in time. 4. Update AIBT with final time
Protocol and activities	CheckFlightLanded, *RequestTaxiOutTime*, *WaitForTaxiOutTime*, ComputeAIBT
Permissions	reads *flight status // (landed)* updates *flight AIBT* *flight status to arrived*
Liveness responsibilities	GAH = (CheckFlightLanded. *RequestTaxiOutTime*.*WaitForTaxiOutTime*.ComputeAIBT)^{w}
Safety responsibilities	true

Table 73. Role schema description Taxi In Provider (TIP)

Role schema	Taxi In Provider (TIP)
Description	Provides the actual taxi in time for a flight
Protocol and activities	ComputeTaxiInTime, *WaitForTaxiInRequest*, *ReturnTaxiInTime*
Permissions	reads *aircraft type* generates *taxi in time*
Liveness responsibilities	TIP = (*WaitForTaxiInRequest*.ComputeTaxiInTime.*ReturnTaxiInTime*)^{w}
Safety responsibilities	true

Table 74. Role schema description Turnaround Operations (TRO)

Role schema	Turnaround Operations (TRO)
Description	When a flight arrives to the gate computes the turnaround time required, generates the ready to depart time of the subsequent flight. 6. Reallocate passengers of arriving flight if needed 7. Computes the turnaround time. 8. Computes delay due to non-ATFM causes 9. Updates the EOBT 10. Requests reassessment of flight departure in case extra delay has been incurred
Protocol and activities	CheckFlightArrival, *RequestProcessArrivalPassengers*, *RequestTurnaroundTime*, *WaitForTurnaroundTime*, ComputeDelayNonDueATFM, GetFollowingFlight, *RequestDepartingReassessment*, *CommunicateWaitForPushbackReady, CommunicateFPtoFligh*
Permissions	reads *flight status // (arrived)* *subsequent flight // subsequent flight of the landed flight* *delay non due to ATFM // all other delay modelled* *subsequent flight EOBT // EOBT of subsequent flight* updates *subsequent flight EOBT* *flight status of arrived flight to finished* *flight status of subsequent flight to ready* generates *turnaround time*

Liveness responsibilities	TRO = (CheckFlightArrival.*(RequestProcessArrivalPassengers* || ProcessFollowigFlihgt))^w ProcessFollowingFlight = GetFollowingFlight.[TurnaroundOperations] TurnaroundOperations =*RequestTurnaroundTime.WaitForTurnaroundTime.* ComputeDelayNonDueATFM.(*RequestDepartingReassessment* |*CommunicateFPtoFlight.CommunicateWaitForPushbackReady*)
Safety responsibilities	▪ Do TurnaroundOperations if there is a next flight ▪ if new EOBT < SOBT then EOBT = SOBT ▪ new EOBT >= EIBT + expected turnaround time

Table 75. Role schema description Airline Passenger Handler (APH)

Role schema	Airline Passenger Handler (APH)
Description	When a flight arrives direct passengers to connecting flights. All passengers connecting to flight that have already left are requested to be rebooked.
Protocol and activities	*WaitForRequestProcessArrivalPassengers*, CheckPassengerMissedConnections, *RequestMinimumConnectingTimes*, *WaitForMinimumConnectingTimes*, *RequestReallocatePax*, AllocatePassengersToNextFlights
Permissions	reads *passenger itineraries // to know connections* *flights and their status // status of flights to know where they are (scheduled, pre-departure, cancelled)* updates *passenger itineraries*
Liveness responsibilities	APH = (*WaitForRequestProcessArrivalPassengers*.CheckPassengerMissedConnections.(*RequestReallocatePax* || (*RequestMinimumConnectingTimes*.*WaitForMinimumConnectingTimes*)+.AllocatePassengersToNextFlights))^w
Safety responsibilities	• If passengers miss their connections, i.e., their outbound flight has already left, the RequestRelocatePax is requested. • passengers who miss connection status = missed • passengers who don't miss connections status = in flight

Table 76. Role schema description Ground Handler (GH)

Role schema	Ground Handler (GH)
Description	Provides the turnaround time for a flight

Protocol and activities	ComputeTurnaroundTime, *WaitForTurnaroundTimeRequest, ReturnTurnaroundTime*
Permissions	reads *aircraft type* *airport type* *airline type* generates *turnaround time*
Liveness responsibilities	GH = (*WaitForTurnaroundTimeRequest*.ComputeTurnaroundTime. *ReturnTurnaroundTime*)^{w}
Safety responsibilities	true

Table 77. Role schema description Departure Reassessment (DR)

Role schema	Departure Reassessment (DR)
Description	When the EOBT of a flight is updated we might have to reassess the ATFM delay and the flight plan of the flight if the flight was already in pre-departure status. 3. If the flight has an ATFM slot and this is missed: request a new ATFM slot and update EOBT. 4. If flight plan could be modified call to RequestFlightPlanRecomputation
Protocol and activities	*WaitForDepartingReassessmentRequest*, *RequestATFMSlot*, *WaitForATFMSlot*, RecomputeEOBT, *RequestFlightPlanRecomputation*, ComputePushBack, *CommicateFPtoFlight*
Permissions	reads *flight status // flight status (scheduled, pre-departure)* *flight EOBT // flights EOBT* *flight COBT // COBT of the flight if it has ATFM delay* updates *ATFM delay of flight* *EOBT of flight* *COBT of flight* *EIBT of flight* generates *PushBack*
Liveness responsibilities	DR = (*WaitForDepartingReassessmentRequest*. [RequestATFMSlot.*WaitForATFMSlot*.RecomputeEOBT]. [*RequestFlightPlanRecomputation*].*CommicateFPtoFlight*.ComputePushBack)^{w}

Safety responsibilities	• If delay too large or flight plan not exist for the flight then RequestFlightPlanRecomputation • If the flight has ATFM delay and EOBT > COBT + 15 minutes a new slot needs to be requested • If ATFM delay is given then COBT = EOBT

Table 78. Role schema description Disseminate Flight Position Update (DFPU)

Role schema	Disseminate Flight Position Update (DFPU)
Description	Updates subscribed entities with position report for flight
Protocol and activities	*CheckFlightCrossingPoint*, *NotifyFlightInEAMAN*
Permissions	reads *entities subscribed to flight crossing point events*
Liveness responsibilities	DFPU = (CheckFlightCrossingPoint.[*NotifyFlightInEAMAN*])^{w}
Safety responsibilities	true

Table 79. Role schema description Flight in AMAN Handler (FIAH)

Role schema	Flight in AMAN Handler (FIAH)
Description	Get notified that a flight has entered/moved in the AMAN and notify the required service from the AMAN
Protocol and activities	*WaitForFlightInEAMAN*, *NotifyFlightInPlanningHorizonEAMAN*, *NotifyFlightInExecutionHorizonEAMAN*
Permissions	-
Liveness responsibilities	FIAH = *WaitForFlightInEAMAN*.(*NotifyFlightInPlanningHorizonEAMAN* | *NotifyFlightInExecutionHorizonEAMAN*))^{w}
Safety responsibilities	true

Table 80. Role schema description Arrival Queue Updater (AQU)

Role schema	Arrival Queue Updater (AQU)

Description	When a flight enters the planning scope of the E-AMAN, its position in the arrival queue is updated. The way the queue is updated is dependent on the FAC implementation: • FAC_0: 'First-in, first-out' approach. First slot available is given to the aircraft. • FAC_1: Reactionary delay is considered when sequencing flights. An estimation of the reactionary delay that will be generated is computed and used in the prioritisation process. • FAC_2: Priorities set by airspace users are considered A message is sent to the flight to update the arrival delay expected.
Protocol and activities	*WaitForFlightInPlanningHorizon*, *RequestFlightArrivalInformation*, *WaitForFlightArrivalInformation*, UpdateArrivalQueue, *UpdateFlightPlanConstraint*
Permissions	reads *arrival queue // with slots in the arrival queue* *flight information* // SIBT, EOBT, Estimated Landing Time (ELT) *next flight rotation times // In FAC_1* *turnaround estimation time // In FAC_1* *taxi in time estimation // In FAC_1* *flight priorities // IN FAC_2* updates *arrival queue by assigning the flight in the queue and marking it as planned*
Liveness responsibilities	AQU = (*WaitForFlightInPlanningHorizon*. [*RequestFlightArrivalInformation*.*WaitForFlightArrivalInformation*]. UpdateArrivalQueue.*UpdateFlightPlanConstraint*)^{w}
Safety responsibilities	• Capacity of arrival runway is maintained • All flights get a position in the arrival queue • Slot provided >= ELT • Slots assigned are not changed • For FAC_1 and FAC_2 RequestFlightArrivalInformation to get priorities and/or information on turnaround estimation times • Flight Plan Constraint = arrival slot

Table 81. Role schema description Flight Arrival Information Provider (FAIP)

Role schema	Flight Arrival Information Provider (FAIP)
Description	Provides information needed to consider the arrival sequencing. • FAC_1: The information provided is related to the reactionary delay • FAC_2: The information provided are the priorities set by airspace users Process might request information from the AOC.
Protocol and activities	*WaitForFlightArrivalInformationRequest*, *RequestFligthArrivalInformationAOC*, *WaitForFlightArrivalInformationAOC*, *ProvideFlightArrivalInformation*
Permissions	reads *flight information* // SIBT, EOBT, Estimated Landing Time (ELT) *flight plan* // information on the flight plan
Liveness responsibilities	FAIP = (WaitForFlightArrivalInformationRequest. [RequestFligthArrivalInformationAOC.WaitForFlightArrivalInformationAOC]. ProvideFlightArrivalInformation)^{w}
Safety responsibilities	• Request Flight Arrival Information from AOC if an update on the priorities is needed in FAC_2 • Request Flight Arrival Information from AOC if an update on the reactionary propagation is needed in FAC_1

Table 82. Role schema description Flight Arrival Information Estimator (FAIE)

Role schema	Flight Arrival Information Estimator (FAIE)
Description	Provides information needed to consider the arrival sequencing from the AOC perspective. • FAC_1: The information provided is related to the reactionary delay • FAC_2: The information provided are the priorities set by airspace user
Protocol and activities	*WaitForFlightArrivalInformationAOCRequest, ProvideFlightArrivalAOCInformation, RequestFlightPrioritisation, WaitForFlightPriorities,* ComputeFlightArrivalInformation
Permissions	reads *flight information* // SIBT, EOBT, Estimated Landing Time (ELT) *flight plan* // information on the flight plan *other AOC flights information* // information on other flights of the AOC

Liveness responsibilities	FAIE = (*WaitForFlightArrivalInformationAOCRequest*. [*RequestFlightPrioritisation.WaitForFlightPriorities*]. ComputeFlightArrivalInformation.*ProvideFlightArrivalAOCInformation*)^{w}
Safety responsibilities	• If FAC_2 it might require the flight priorities to be recomputed with most updated information • If FAC_2 provides the priorities of the flights • If FAC_1 provides information on the turnaround and estimated reactionary delay

Table 83. Role schema description Arrival Slot Provider (ASP)

Role schema	Arrival Slot Provider (ASP)
Description	When a flight enters the execution horizon the slot is assigned to the flight, thus fixing the arrival queue.
Protocol and activities	*WaitForFlightInExectionHorizon*, UpdateArrivalQueue, *UpdateFlightPlanConstraint*
Permissions	reads *arrival queue // with slots in the arrival queue* *flight information // SIBT, EIBT, Estimated Landing Time (ELT)* updates *arrival queue by assigning the flight in the queue and marking it as assigned*
Liveness responsibilities	ASP = (*WaitForFlightInExectionHorizon*.UpdateArrivalQueue. *UpdateFlightPlanConstraint*)^{w}
Safety responsibilities	• Capacity of arrival runway is maintained • Slot provided >= ELT • Flight Plan Constraint = arrival slot

Table 84. Role schema description Flight Plan Constraint Updater (FPCU)

Role schema	Flight Plan Constraint Updater (FPCU)
Description	Updates constraints of flight plan of a flight.
Protocol and activities	*WaitForFlightPlanConstraintUpdateRequest, UpdateFlightPlanConstraint*
Permissions	Updates *flight plan constraints*
Liveness responsibilities	FPCU = (*WaitForFlightPlanConstraintUpdateRequest*.UpdateFlightPlanConstraint)^{w}

Safety responsibilities	▪ If constraint does not exists add it to constraints to trajectories ▪ If constraint exist update it with new constraint.

2 Annex – Interactions model

2.1 Airline Flight Planner interactions

a) Request ATFM slot

Table 85. Interaction model Request ATFM slot

RequestATFMSlot	
Initiator	**Responder**
Air Flight Planner (AFP)	Network Manger Flight Plan Processing (NMFPP)
Inputs	
None	
Processing	
Request an ATFM slot for a given flight	
Outputs	
flight plan	

Table 86. Interaction model Return ATFM delay

ReturnATFMDelay	
Initiator	Responder
Network Manager Flight Plan Processing (NMFPP)	Air Flight Planner (AFP)
Inputs	
flight plan	
Processing	
Return delay assigned by NM	
Outputs	
ATFM delay	

b) Request cost of options of FP modifications

Table 87. Interaction model Request FP Cost Options Estimation

RequestFPCostOptionsEstimation	
Initiator Air Flight Planner (AFP)	**Responder** Airline Pre-departure Flight Plan cost Computer (AFPC)
Inputs None	
Processing Request an FP options with their associated cost	
Outputs flight id	

Table 88. Interaction model Return FP Cost per Option

ReturnFPCostPerOption	
Initiator Airline pre-departure Flight Plan cost Computer (AFPC)	**Responder** Air Flight Planner (AFP)
Inputs flight id	
Processing Compute the best option for each action and their associated costs Return estimation of costs of each option with their associated actions	
Outputs Wait for passenger flight plans with cost Flight modification with cost Cost of cancellation option	

c) Request flight prioritisation

Table 89. Interaction model Request Flight Prioritisation

RequestFlightPrioritisation	
Initiator	**Responder**
Air Flight Planner (AFP)	Airline pre-departure flight prioritiser (ADFP)

Inputs	
None	
Processing	
Request the prioritisation of a flight	
Outputs	
flight id	

Table 90. Interaction model Return Flight Priority

ReturnFlightPriority	
Initiator	**Responder**
Airline pre-departure flight prioritiser (ADFP)	Air Flight Planner (AFP)
Inputs	
flight id	
Processing	
Return the flight priority and the credits if FP_2	
Outputs	
flight priority	
flight credit (FP_2)	

d) Request flight swap

Table 91. Interaction model Flight Swap Request

FlightswapRequest	
Initiator	**Responder**
Air Flight Planner (AFP)	Flight Swapper (FS)
Inputs	
None	
Processing	
Request the prioritisation of a flight or a set of flights	
Outputs	

flight(s) id(s)

flight(s) priority(ies)

flight(s) credit(s) (FP_2)

Table 92. Interaction model Return Flight Swap

ReturnFlightSwap	
Initiator	**Responder**
Flight Swapper (FS)	Air Flight Planner (AFP)
Inputs	
flight id	
flight priority	
flight credit (FP_2)	
Processing	
Swap flights if available and return new list of flight delays	
Outputs	
flights delays	

e) Accept flight plan

Table 93. Interaction model Accept FP

AcceptFP	
Initiator	**Responder**
Air Flight Planner (AFP)	Network Manager Accept and Disseminate FP (NMAD)
Inputs	
None	
Processing	
Request the acceptance of a flight plan	
Outputs	
flight plan with notification points	

Table 94. Interaction model Return Point Notification FP

ReturnPointsNotificationFP	
Initiator	**Responder**
Network Manager Accept and Disseminate FP (NMAD)	Air Flight Planner (AFP)
Inputs	
flight plan	
Processing	
Disseminate the flight plan	
Return the notification points for the flight plan	
Outputs	
flight plan with notification points	

Table 95. Interaction model Request Dissemination of FP

RequestDisseminationOfFP	
Initiator	**Responder**
Network Manager Accept and Disseminate FP (NMAD)	Disseminate FP (DFP)
Inputs	
flight plan	
Processing	
Request the dissemination of a flight plan	
Outputs	
flight plan with notification points	

Table 96. Interaction model Update Planned Arrival of FP

UdpatePlannedArrivalFP	
Initiator	**Responder**
Disseminate FP (DFP)	Arrival Queue Planned Updater (AQPU)
Inputs	
flight id, EIBT | cancelled	
Processing	

Request the update of the position of a flight in the arrival queue by updating its EIBT or indicating the flight has been cancelled

Outputs

None

Table 97. Interaction model Return Request Points Notification

ReturnRequestPointsNotification	
Initiator	**Responder**
Arrival Queue Planned Updater (AQPU)	Disseminate FP (DFP)

Inputs

flight plan

Processing

Provide the flight plan with the points where to notify

Outputs

flight plan with notification points

Table 98. Interaction model Update Planned Departure FP

UpdatePlannedDepartureFP	
Initiator	**Responder**
Disseminate FP (DFP)	Departure Queue Updater (DQU)

Inputs

flight id, EOBT | cancelled

Processing

Request the update of the position of a flight in the departing queue by updating its EOBT or indicating the flight has been cancelled

Outputs

None

Table 99. Interaction model Communicate FP to Flight

CommunicateFPtoFlight	
Initiator	**Responder**
Airline Flight Planner (AFP)	Flight Plan Updater (FPU)
Inputs	
None	
Processing	
Communicate flight plan information to be used by a given flight (route, speeds, waiting-for-passengers)	
Outputs	
flight plan with notification points	

f) Cancel Flight Plan

Table 100. Interaction model Communicate FP to Flight

CommunicateFPtoFlight	
Initiator	**Responder**
Airline Flight Planner (AFP)	Flight Plan Updater (FPU)
Inputs	
None	
Processing	
Cancel the flight plan	
Outputs	
flight plan cancelled	

Table 101. Interaction model Cancel FP

Cancel FP	
Initiator	**Responder**
Air Flight Planner (AFP)	Network Manager Cancel FP (NMC)
Inputs	
None	

Processing

Request the cancellation of a flight plan

Outputs

None

Table 102. Interaction model Request Dissemination Cancellation FP

RequestDisseminationCancellationFP	
Initiator	**Responder**
Network Manager Cancel FP (NMC)	Disseminate Cancellation FP (DCFP)
Inputs	
flight id	
Processing	
Disseminate cancellation of flight plan.	
Outputs	
None	

Table 103. Interaction model Cancellation Arrival FP

CancellationArrivalFP	
Initiator	**Responder**
Disseminate Cancellation FP (DCFP)	Arrival Queue Planned Updater (AQPU)
Inputs	
flight id	
Processing	
Remove flight from planned arrival	
Outputs	
None	

Table 104. Interaction model Cancellation Departure FP

CancellationDepartureFP	
Initiator	**Responder**
Disseminate Cancellation FP (DCFP)	Departure Queue Updater (DQU)
Inputs	
flight id	
Processing	
Remove flight from planned departure	
Outputs	
None	

g) Communicate flight ready to wait for push back

Table 105. Interaction model Communicate Wait for Pushback Ready

CommunicateWaitForPushbackReady	
Initiator	**Responder**
Airline Flight Planner (AFP)	Wait for Pushback_Ready (WFBR)
Inputs	
None	
Processing	
wait for push_back ready event	
Outputs	
None	

h) Request reallocation of passengers

Table 106. Interaction model Passenger Reallocation Request

PassengerReallocationRequest	
Initiator	**Responder**
Air Flight Planner (AFP)	Passenger Reallocation (PR)

Inputs

flight from where to be reallocated

passengers itineraries to be reallocated

Passenger types to be reallocated

Processing

Request the reallocation of passengers itineraries from a given flight.

Recompute itineraries and reallocate them to other flights or book them in hotels and cancel the itineraries

Outputs

None

Table 107. Interaction model Request Minimum Connecting Times

RequestMinimumConnectingTimes	
Initiator	**Responder**
Passenger Reallocation (PR)	Provide Connecting Times (PCT)

Inputs

passengers itineraries types and airports

Processing

Request the connecting time for a set of itineraies

Outputs

None

Table 108. Interaction model Return Connecting Times

ReturnConnectingTimes	
Initiator	**Responder**
Provide Connecting Times (PCT)	Passenger Reallocation (PR)

Inputs

None

Processing	
Compute connecting times for passengers	
Outputs	
Connecting times for the passengers itineraries	

Table 109. Interaction model Request Passengers Preference Rebooking

RequestPaxPreferenceRebooking	
Initiator	**Responder**
Passenger Reallocation (PR)	Passenger Preference in Connection (PPC)
Inputs	
Passenger options missing connections	
Processing	
Request the preferences of passengers for different rebooking options	
Outputs	
passengers itineraries	

Table 110. Interaction model Return Passenger Preferences in Connection

ReturnPassengerPreferencesConnection	
Initiator	**Responder**
Passenger Preference in Connection (PPC)	Passenger Reallocation (PR)
Inputs	
passenger itineraries options	
Processing	
Preferences per option for the passengers itineraries	
Outputs	
passenger itineraries options ranked	

2.2 Airline Turnaround Operations

a) Request reassessment of departing

Table 111. Interaction model Request Departing Reassessment

RequestDepartingReassessment	
Initiator	**Responder**
Airline Turnaround Operations (TRO)	Departure Reaessment (DR)
Inputs	
None	
Processing	
Request the reassessment of a departure	
Outputs	
None	

b) Request process arrival passengers

Table 112. Interaction model Request Process Arrival Passengers

RequestProcessArrivalPassengers	
Initiator	**Responder**
Airline Turnaround Operations (TRO)	Airline Passenger Handler (APH)
Inputs	
None	
Processing	
Request processing of arrival passengers with follow up flights	
Outputs	
passenger itineraries with connections	

Table 113. Interaction model Request Minimum Connecting Times

RequestMinimumConnectingTimes	
Initiator	**Responder**
Airline Passenger Handler (APH)	Provide Connecting Times (PCT)
Inputs	
passengers itineraries types and airports who hasn't missed connection yet	
Processing	
Request the connecting time for a set of itineraies	
Outputs	
None	

Table 114. Interaction model Return Connecting Times

ReturnConnectingTimes	
Initiator	**Responder**
Provide Connecting Times (PCT)	Airline Passenger Handler (APH)
Inputs	
None	
Processing	
Compute connecting times for passengers	
Outputs	
Connecting times for the passengers itineraries	

Table 115. Interaction model Request Reallocate Passengers

RequestReallocatePax	
Initiator	**Responder**
Airline Passenger Handler (APH)	Air Flight Planner (AFP)
Inputs	
itineraries passengers with missed connections	
Processing	
Request passenger missed connections reallocated	
Outputs	
None	

c) Aircraft turnaround operations

Table 116. Interaction model Request Turnaround Time

RequestTurnaroundTime	
Initiator	**Responder**
Airline Turnaround Operations (TRO)	Ground Handler (GH)
Inputs	
aircraft type, airline type, airport	
Processing	
Request turnaround time for a given aircraft	
Outputs	
None	

Table 117. Interaction model Return Turnaround Time

ReturnTurnaroundTime	
Initiator	**Responder**
Ground Handler (GH)	Airline Turnaround Operations (TRO)
Inputs	
None	
Processing	
Compute turnaround time for aircraft	
Outputs	
turnaround time	

Table 118. Interaction model Communicate FP to Flight

CommunicateFPtoFlight	
Initiator	**Responder**

Airline Turnaround Operations (TRO)	Flight Plan Updater (FPU)
Inputs	
None	
Processing	
Communicate flight plan information to be used by a given flight (route, speeds, waiting-for-passengers)	
Outputs	
flight plan with notification points	

Table 119. Interaction model Communicate Wait For Pushback Ready

CommunicateWaitForPushbackReady	
Initiator	**Responder**
Airline Turnaround Operations (TRO)	Wait for Pushback_Ready (WFBR)
Inputs	
None	
Processing	
wait for push_back ready event	
Outputs	
None	

2.3 Departure reassessment

a) Request ATFM slot

Table 120. Interaction model Request ATFM Slot

RequestATFMSlot	
Initiator	**Responder**
Departure Reassessment (DR)	Network Manager Flight Plan Processing (NMFPP)
Inputs	

None

Processing

Request an ATFM slot for a given flight

Outputs

flight plan

Table 121. Interaction model Return ATFM Slot

ReturnATFMSlot	
Initiator	**Responder**
Network Manager Flight Plan Processing (NMFPP)	Departure Reassessment (DR)
Inputs	
flight plan	
Processing	
Return delay assigned by NM	
Outputs	
ATFM delay	

b) Request recomputation of flight plan

Table 122. Interaction model Request Flight Plan Recomputation

RequestFlightPlanRecomputation	
Initiator	**Responder**
Departure Reassessment (DR)	Air Flight Planner (AFP)
Inputs	
flight plan	
Processing	
Request the flight plan to be recomputed	
Outputs	

None

c) Communicate flight plan update

Table 123. Interaction model Communicate FP to Flight

CommunicateFPtoFlight	
Initiator	**Responder**
Departure Reassessment (DR)	Flight Plan Updater (FPU)
Inputs	
None	
Processing	
Communicate flight plan information to be used by a given flight (route, speeds, waiting-for-passengers)	
Outputs	
flight plan with notification points	

2.4 Flight Arrival Information Estimator

a) Request flight prioritisation

Table 124. Interaction model Request Flight Prioritisation

RequestFlightPrioritisation	
Initiator	**Responder**
Flight Arrival Information Estimator (FAIE)	Airline pre-departure flight prioritiser (ADFP)
Inputs	
None	
Processing	
Request the prioritisation of a flight	
Outputs	
flight id	

Table 125. Interaction model Return Flight Prioritisation

ReturnFlightPriority	
Initiator	**Responder**
Airline pre-departure flight prioritiser (ADFP)	Flight Arrival Information Estimator (FAIE)
Inputs	
flight id	
Processing	
Return the flight priority and the credits if FP_2	
Outputs	
flight priority	
flight credit (FP_2)	

2.5 Aircraft Departing Handler

a) Request taxi out time

Table 126. Interaction model Request Taxi Out Time

RequestTaxiOutTime	
Initiator	**Responder**
Aircraft Departing Handler (ADH)	Taxi Out Provider (TOP)
Inputs	
flight	
airport	
Processing	
Request the time required to do the taxi out for a given flight	
Initiator	
Aircraft Departing Handler (ADH)	

Table 127. Interaction model Return Taxi Out Time

ReturnTaxiOutTime	
Initiator	**Responder**
Taxi Out Provider (TOP)	Aircraft Departing Handler (ADH)
Inputs	
None	
Processing	
Compute the taxi out time	
Outputs	
taxi out time	

2.6 Departing Slot Requester

a) Request taxi out estimation

Table 128. Interaction model Request Taxi Out Time Estimation

RequestTaxiOutTimeEstimation	
Initiator	**Responder**
Aircraft Departing Handler (ADH)	Taxi Out Estimator (TOE)
Inputs	
flight	
airport	
Processing	
Request the estimated time required to do the taxi out for a given flight	
Outputs	
None	

Table 129. Interaction model Return Taxi Out Time Estimation

ReturnTaxiOutTimeEstimation	
Initiator	**Responder**
Taxi Out Estimator (TOE)	Aircraft Departing Handler (ADH)

Inputs

None

Processing

Estimate the taxi out time

Outputs

estimated taxi out time

b) Request departing slot

Table 130. Interaction model Request Departing Slot

RequestDepartingSlot	
Initiator	**Responder**
Departing Slot Requester (DSR)	Departure Slot Provider (DSP)
Inputs	
flight id	
Processing	
Request a departure slot when flight is ready.	
Outputs	
None	

Table 131. Interaction model Return Departure Time

ReturnDepartureTime	
Initiator	**Responder**
Departure Slot Provider (DSP)	Departing Slot Requester (DSR)
Inputs	
None	
Processing	
Return the departing slot for the flight	

Outputs	
Departing slot time	

2.7 Ground Arrival Handler

a) Request taxi in time

Table 132. Interaction model Request Taxi In Time

RequestTaxiInTime	
Initiator	**Responder**
Ground Arrival Handler (GAH)	Taxi In Provider (TIP)
Inputs	
flight	
airport	
Processing	
Request the time required to do the taxi in for a given flight	
Outputs	
None	

Table 133. Interaction model Return Taxi In Time

ReturnTaxiInTime	
Initiator	**Responder**
Taxi In Provider (TIP)	Ground Arrival Handler (GAH)
Inputs	
None	
Processing	
Compute the taxi in time	
Outputs	
taxi in time	

2.8 Operate Trajectory

a) Communicate update EIBT

Table 134. Interaction model Communicate Update EIBT

CommunicateUpdateEIBT	
Initiator	**Responder**
Operate Trajectory (OT)	EIBT Updater (EU)
Inputs	
flight id, EIBT	
Processing	
EIBT has been estimated and need to be updated	
Outputs	
None	

2.9 Disseminate Flight Position Update

a) Disseminate flight has moved within the E-AMAN

Table 135. Interaction model Notify Flight In EAMAN

NotifyFlihgtInEAMAN	
Initiator	**Responder**
Disseminate Flight Position Update (DFPU)	Flight in AMNA Handler (FIAH)
Inputs	
None	
Processing	
Notify flight has reached a significant point within the EAMAN	
Outputs	
None	

2.10 Flight in AMAN Handler

a) Notify E-AMAN flight has entered Planning Horizon

Table 136. Interaction model Notify Flight In Planning Horizon

NotifyFlihgtInPlanningHorizon	
Initiator	**Responder**
Flight in AMNA Handler (FIAH)	Arrival Queue Updater (AQU)
Inputs	
flight entering planning horizon	
Processing	
Notify flight has reached planning horizon and needs update in arrival queue	
Outputs	
None	

b) Notify E-AMAN flight as entered Execution Horizon

Table 137. Interaction model Notify Flight In Execution Horizon

NotifyFlihgtInPlanningHorizon	
Initiator	**Responder**
Flight in AMNA Handler (FIAH)	Arrival Slot Provider (ASP)
Inputs	
flight entering execution horizon	
Processing	
Notify flight has reached execution horizon and needs final arrival slot assigned	
Outputs	
None	

2.11 Arrival Queue Updater

a) Request flight arrival information to update arrival slot

Table 138. Interaction model Request Flight Arrival Information

RequestFlightArrivalInformation	
Initiator	**Responder**

Arrival Queue Updater (AQU)	Flight Arrival Information Provider (FAIP)

Inputs

None

Processing

Request further information from flight if FAC_2

Outputs

None

Table 139. Interaction model Request Flight Arrival Information AOC

RequestFlightArrivalInformationAOC	
Initiator	**Responder**
Flight Arrival Information Provider (FAIP)	Flight Arrival Information Estimator (FAIE)

Inputs

EIBT

Processing

Request information of priorities for flight in E-AMAN

Outputs

None

Table 140. Interaction model Provide Flight Arrival AOC Information

ProvideFlightArrivalAOCInformation	
Initiator	**Responder**
Flight Arrival Information Estimator (FAIE)	Flight Arrival Information Provider (FAIP)

Inputs

None

Processing

Compute information of priorities for flight in E-AMAN

Outputs

Flight prioririte s for E-AMAN

credits (FP_2)

Table 141. Interaction model Provide Flight Arrival Information

ProvideFlightArrivalInformation	
Initiator	**Responder**
Flight Arrival Information Provider (FAIP)	Arrival Queue Updater (AQU)
Inputs	
Flight priorities for E-AMAN	
Processing	
None	
Outputs	
priorities (FP_2)	
other information (FP_1)	

b) Provide updated arrival slot to flight

Table 142. Interaction model Update Flight Plan Constraint

UpdateFlightPlanConstraint	
Initiator	**Responder**
Arrival Queue Updater (AQU)	Flight Plan Constraint Updater (FPCU)
Inputs	
flight	
slot assigned at arrival	
Processing	
Assign arrival slot to flight and communicate that arrival constraint	
Outputs	
None	

2.12 Arrival Slot Provider

a) Provide arrival slot to flight

Table 143. Interaction model Update Flight Plan Constraint

UpdateFlightPlanConstraint	
Initiator	**Responder**
Arrival Slot Provider (ASP)	Flight Plan Constraint Updater (FPCU)
Inputs	
flight	
slot assigned at arrival	
Processing	
Assign final arrival slot to flight and communicate that arrival constraint	
Outputs	
None	

3 Annex – Services model

Table 144. Service model for Airline Operating Centre Agent

Service	Inputs	Outputs	Pre-condition	Post-condition
Check flights are ready for flight plan submission	* Event FP_submission	-	FP_submission = true	true
Check flight has pushed back	* Event Pushback	-	Pushback = true	true
Check flight has arrived	* Event Flight_Arrival	-	Flight_Arrival = true	true
Update flight plan information	flight plan information	flight plan updated, ATFM delay	true	true
Request ATFM slot	flight plan information	ATFM delay	flight plan defined	if ATFM delay then CTOT updated and EOBT = CTOT
Compute flight plan options cost estimation	flight information	flight plan options and their associated cost	true	true
Choose flight plan option	flight plan options	flight plan selected	if 4DT_1 or 4DT_2 flight plan cost options requested	if 4DT_0 rule of thumb applied to select option if 4DT_1 or 4DT2 option with best expected outcome selected
Request flight prioritisation	flight information	flight priority	FP_1 or FP_2	true
Request flight swap	flight priorities	flight ATFM delay	FP_1 or FP_2 and priorities indicate swap is selected	flights involved in swap updated ATFM delay

Recompute EOBT	flight plan and ATFM delay	EOBT update	true	EOBT updated considering ATFM delay and flight plan
Recompute EIBT	flight plan and EOBT	EIBT update	EOBT estimated for flight plan and ATFM delay	EIBT = EOBT + flight plan duration (including taxi times estimations)
Accept flight plan	flight plan	flight plan with notification points	flight plan defined and ATFM slot requested	flight plan has notification points
Cancel flight plan	flight id	-	flight plan defined	passengers reaccommodated
Communicate wait for pushback to flight	flight id	-	flight plan accepted and flight plan information updated	true
Update EOBT from EIBT change	flight f1, updated EIBT	updated flight plan of next flight of f1 (f2)	flight status = {scheduled, pre-departure}	if new EOBT < SOBT then EOBT = SOBT new EOBT >= EIBT + expected turnaround time new EOBT >= COBT if status of f2 is pre-departure then Request reassessment of departure
Obtain following flight	flight (f1)	flight (f2) next rotation of (f1) or null	true	f2 is the next flight of aircraft in flight f1 null if f1 is the last flight of the aircraft in f1
Departure reassessment	flight, EOBT, COBT	updated flight plan and times for flight	flight status = {scheduled, pre-departure}	if (COBT is not null) and (ATFM delay + EOBT > COBT +15) then new ATFM delay requested
Compute passenger missing connections	passengers itineraries, minimum connecting times, flights EOBT	passengers itineraries which missed connections	true	passengers with arrival at gate (arrival time + MCT) > EOBT of follow flight on

				their itinerary allocated passengers to flights with arrived time of passenger + connecting time <= EOBT of next flight. Passenger itineraries status = {in flight} Request reallocation of passengers with arrived time of passenger + connecting time > EOBT of next flight. passenger itineraries status = {missed}
Estimate information for arrival coordination of a flight	flight, EIBT	information of flights estimated delays (FAC_1) priorities for flight arrival (FAC_2)	flight in the horizon of the arrival coordinator	In FAC_2 priorities could be updated
Request connecting times for passengers	passengers itineraries airport	connecting times for passengers	true	true
Request priorities for passengers rebooking	passenger itineraries missing connections passengers itineraries options	priorities for rebooking of passengers	true	true
Check load factors of flights	flights and passengers itineraries	load factors of flights with seats available	true	true
Compute turnaround of flight	flight f1	updated flight plan of next flight of f1 (f2)	flight f1 arrived	f2 times updated considering turnaround time required
Request turnaround time for flight	flight airport	turnaround time	true	true

Table 145. Service model for Flight Agent

Service	Inputs	Outputs	Pre-condition	Post-condition
Check push back ready event arrives	* Event Pushback_Ready	-	Pushback_Ready = true	true
Check push back event arrives	* Event Pushback	-	Pushback = true	true
Check Takeoff event arrives	* Event Takeoff	-	Takeoff = true	true
Check Flight crosses significant point	* Event Flight_Crossing_Point	-	Flight_Crossing_Point = true	true
Check flight has landed	* Event Landed	-	Landed = true	true
Request taxi in time	flight airport	taxi in time	Landed = true	true
Request taxi out estimation	flight airport	taxi out time estimation	Pushback_Ready = true	true
Request taxi out time	flight airport	taxi out time	Pushback = true	true
Compute flight segment	flight	flight trajectory until reporting point	flight after takeoff before landing	true
Compute delay non due to ATFM	flight	delay	true	true
Update flight plan information	new flight plan information	-	flight status different to {landed}	flight plan information for flight = new flight plan information
Update flight plan constraint	flight plan constraint (e.g. landing time)	-	flight plan defined	flight plan constraint considered in recomputation of trajectory

Request flight arrival information from AOC	flight info	flight arrival information	flight status = {airborne}	flight information from AOC received
Provide flight arrival information	flight arrival information	-	flight status = {airborne}	true
Wait for pushback ready	-	-	flight status = {pre-departure}	true
Communicate EIBT EIBT update		-	true	true
Update EIBT	flight plan	-	true	EIBT computed considering flight plan and taxi in estimation
Compute take off time	AOBT taxi out	-	true	take-off = AOBT + taxi out
Compute AOBT	pushback_ready departing slot taxi out estimation	-	true	AOBT = departing slot - taxi out estimation
Request departing slot	flight, EOBT	departing slot departing delay	flight status = {ready}	true

Table 146. Service model for Radar Agent

Service	Inputs	Outputs	Pre-condition	Post-condition
Check Flight crosses significant point	* Event Flight_Crossing_Point	-	Flight_Crossing_Point = true	true
Disseminate flight crossing point update	flight crossing point	-	Flight_Crossing_Point = true	agents interested in being notified of flight crossing point notified
Disseminate flight plan	flight plan	flight plan with notification points	true	flight plan reported to E-AMAN and DMAN points of interest reported
Disseminate cancellation of flight plan	flight id	-	true	flight cancellation reported to E-AMAN and DMAN

Table 147. Service model for Network Manager Agent

Service	Inputs	Outputs	Pre-condition	Post-condition
Flight plan processing	flight	ATFM delay	flight_status = {pre-departure}	capacities maintained pre-tactically
Accept and disseminate flight plan	flight plan	flight plan with notification point	true	true
Cancel flight plan	flight id	-	flight accepted previously	true

Table 148. Service model for Ground Airport Agent

Service	Inputs	Outputs	Pre-condition	Post-condition
Provide connecting times	passengers itineraries airport	connecting times	true	true
Provide taxi out estimation	flight airport	taxi out time estimation	true	true
Provide taxi out	flight airport	taxi out time	true	true
Provide taxi in	flight airport	taxi in time	true	true
Provide turnaround time	flight airport	turnaround time	true	true

Table 149. Service model for Flight Prioritisation Agent

Service	Inputs	Outputs	Pre-condition	Post-condition
Compute flight swap	flights and their priorities flight credits (FP_2)	flights and their delay	flights have delay and priorities	capacities are maintained

Table 150. Service model for Passenger Agent

Service	Inputs	Outputs	Pre-condition	Post-condition
Provide passengers preferences in connection	passenger itineraries options	passengers preferences for options	passengers have missed connection	all options ranked by preference

Table 151. Service model for E-AMAN Agent

Service	Inputs	Outputs	Pre-condition	Post-condition
Build arrival queue	flight schedules arrival capacity	arrival queue	true	arrival queue slots meet arrival capacity all flight in arrival queue status = {scheduled}
Update planned arrival queue	flight, EIBT, taxi in estimation	-	flight in arrival queue flight status = {scheduled}	capacity in queue not changed slot assigned to flight >= EIBT - estimated taxi in slots with flights with status = {assigned} not changed return points where to be notified for flight
Request arrival information to flight	flight	flight arrival information	flight status = {in_FAC}	true
Update arrival queue			flight status = {in_FAC}	
Communicate arrival slot to flight with updated flight plan constraint	flight, arrival slot	-	flight status = {in_FAC, in_execFAC}	if flight status = {in_FAC} then slot status = {assigned planning} if flight status = {in_execFAC} then slot status = {assigned}
Handle flight in EMAN	flight position	-	flight status = {in_FAC, in_execFAC}	Planned arrival queue or arrival queue updated.

Table 152. Service model for DMAN Agent

Service	Inputs	Outputs	Pre-condition	Post-condition
Build departure queue	flight schedules departure capacity	departure queue	true	departure queue slots meet departure capacity all flights departure queue status = {scheduled}
Update planned departure queue	flight, EOBT, taxi out estimation	-	flight in departure queue flight status = {scheduled, pre-	capacity in queue not changed slot assigned to flight >= EOBT + estimated taxi out slots with flights with status = {assigned} not changed

			departure}	slot in status = {assigned planning}
Provide departing slot	flight, ready time at gate (RDY), taxi out estimation	departing slot, delay	flight status = {ready}	capacity in queue not changed slot assigned to flight >= RDY + estimated taxi out

-END OF DOCUMENT-