|  |  |
| --- | --- |
| SESAR_JU | |
| E.02.14-M07-DCI-4HD2D-Platform Extension Report 2 | |
| Document information | |
| Project Title | DCI-4HD2D |
| Project Number | E.02.14 |
| Project Manager | The Innaxis Foundation and Research Institute |
| Deliverable Name | Platform Extension Report 2 |
| Deliverable ID | M07 (D2.2 in contract) |
| Edition | 01.00.00 |
| Template Version | 03.00.00 |
| Task contributors | |
| The Innaxis Foundation and Research Institute and the University of Westminster. | |
|  | |
| Abstract  This document is the final report detailing the CASSIOPEIA platform extension carried out in the DCI-4HD2D project. The document covers the agent-based extension design and its corresponding implementation details. This report updates the previous deliverable M06/D2.1 Platform extension intermediate report. | |

Authoring & Approval

|  |  |  |
| --- | --- | --- |
| Prepared By - Authors of the document. | | |
| Name & Company | Position & Title | Date |
| Jorge Martín / The Innaxis Foundation and Research Institute | Consortium Member | 28/05/2016 |
| Luis Delgado / University of Westminster | Consortium Member | 28/05/2016 |
| Alberto Blanch / The Innaxis Foundation and Research Institute | Consortium Member | 28/05/2016 |

|  |  |  |
| --- | --- | --- |
| Reviewed By - Reviewers internal to the project. | | |
| Name & Company | Position & Title | Date |
| David Pérez / The Innaxis Foundation and Research Institute | Consortium Member | 31/05/2016 |
| Samuel Cristóbal / The Innaxis Foundation and Research Institute | Consortium Member | 31/05/2016 |

|  |  |  |
| --- | --- | --- |
| Reviewed By - Other SESAR projects, Airspace Users, staff association, military, Industrial Support, other organisations. | | |
| Name & Company | Position & Title | Date |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| Approved for submission to the SJU By - Representatives of the company involved in the project. | | |
| Name & Company | Position & Title | Date |
| David Pérez / The Innaxis Foundation and Research Institute | Consortium Member | 01/06/2016 |

|  |  |  |
| --- | --- | --- |
| Rejected By - Representatives of the company involved in the project. | | |
| Name & Company | Position & Title | Date |
|  |  |  |

|  |
| --- |
| Rational for rejection |
|  |

Document History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Edition | Date | Status | Author | Justification |
| 00.00.01 | 28/05/2016 | First Version | Jorge Martín | New Document |

Intellectual Property Rights (foreground)

Foreground owned by one or several Members or their Affiliates.

**Table of contents:**

[1 Introduction 5](#_Toc452633900)

[1.1 Purpose 5](#_Toc452633901)

[1.2 Intended Readership 5](#_Toc452633902)

[1.3 Structure 5](#_Toc452633903)

[1.4 Methodology 5](#_Toc452633904)

[1.5 Context 6](#_Toc452633905)

[1.6 Acronyms and Terminology 6](#_Toc452633906)

[2 Software Architecture Extension 8](#_Toc452633907)

[2.1 Command Line Execution 9](#_Toc452633908)

[2.2 Automate Output 11](#_Toc452633909)

[3 Simulation Dataset 14](#_Toc452633910)

[3.1 Extend Flight Dataset with Pre-Computation 15](#_Toc452633911)

[3.2 Update BADA Datasets 17](#_Toc452633912)

[4 Dynamic Cost Indexing 20](#_Toc452633913)

[4.1 DCI Optimisation 21](#_Toc452633914)

[4.2 DCI Strategies 25](#_Toc452633915)

[4.3 DCI Costs 26](#_Toc452633916)

[5 Delay 29](#_Toc452633917)

[5.1 In Block Delay 29](#_Toc452633918)

[5.2 Taxi Uncertainty Values 30](#_Toc452633919)

[5.3 Airborne Uncertainties 31](#_Toc452633920)

[6 Arrival and Departure Management 32](#_Toc452633921)

[6.1 DMAN: FIFO Approach 32](#_Toc452633922)

[6.2 AMAN: Arrival by Priority 33](#_Toc452633923)

[7 Validation & Analysis 35](#_Toc452633924)

[7.1 Validate DCI Speed Selection 36](#_Toc452633925)

[7.2 Validate Flight Simulation 39](#_Toc452633926)

[7.3 Scenario Analysis 39](#_Toc452633927)

[8 Deployment 42](#_Toc452633928)

[8.1 Requirements 42](#_Toc452633929)

[8.2 Step-by-step guide. 42](#_Toc452633930)

[Appendix A – CSV Cost Output File 43](#_Toc452633931)

**List of Figures**

[Figure 2.1 CASSIOPEIA simulation software architecture 8](#_Toc452633706)

[Figure 2.2 CASSIOPEIA simulation software workflow 9](#_Toc452633707)

[Figure 2.3 List of activities performed during simulation execution 11](#_Toc452633708)

[Figure 2.4 List of activities performed during CSV export 13](#_Toc452633709)

[Figure 3.1 Simulation dataset database model 14](#_Toc452633710)

[Figure 3.2 Simulation dataset class model 15](#_Toc452633711)

[Figure 3.3 Flight table comparison 16](file:///C:\Users\ablanch\Downloads\E.02.14-D3.2-DCI-4HD2D-Platform%20Extension.docx#_Toc452633712)

[Figure 3.4 Structure of BADA tables 18](#_Toc452633713)

[Figure 3.5 Aircraft data classes 19](#_Toc452633714)

[Figure 4.1 Airline agent structure 20](#_Toc452633715)

[Figure 4.2 DCI optimization classes 20](#_Toc452633716)

[Figure 4.3 Outbound flight DCI processes on the ground 21](#_Toc452633717)

[Figure 4.4 Inbound flight DCI processes on top of climb 22](#_Toc452633718)

[Figure 4.5 Outbound flight DCI processes on top of climb 22](#_Toc452633719)

[Figure 4.6 Class structure for DCI strategies 26](#_Toc452633720)

[Figure 4.7 Passenger costs class model 27](#_Toc452633721)

[Figure 4.8 Non-passenger cost class model 28](#_Toc452633722)

[Figure 5.1 Initial delay generation for flights 29](#_Toc452633723)

[Figure 5.2 Delay enum type 30](#_Toc452633724)

[Figure 5.3 Agent attribute database model 30](#_Toc452633725)

[Figure 5.4 Airport persistence model 31](#_Toc452633726)

[Figure 6.1 Airport agent structure 32](#_Toc452633727)

[Figure 6.2 Calculate DMAN delay approach 33](#_Toc452633728)

[Figure 6.3 AMAN delay negotiation 34](#_Toc452633729)

[Figure 7.1 Output database model 35](#_Toc452633730)

[Figure 7.2 Output persistence model 36](#_Toc452633731)

[Figure 7.3 Hierarchical model of proposal classes 37](#_Toc452633732)

[Figure 7.4 Decision class in persistence model 37](#_Toc452633733)

[Figure 7.5 Decision table in database model 38](#_Toc452633734)

[Figure 7.6 List of activities in scenario analysis 40](#_Toc452633735)

[Figure 7.7 Interactions between analysis classes 41](#_Toc452633736)

List of tables

[Table 1 Cost estimation and flight phases 24](#_Toc452633845)

# Introduction

## Purpose

This document describes the enhancements to the CASSIOPEIA simulation platform to fully cover the DCI-4HD2D case study. The deliverable documents the enhancements in all layers of the simulator architecture, both functional and conceptual, and the overall process of adding, extending or changing different components of the CASSIOPEIA simulation platform. Thus, it is of interest for any potential user of the platform including policymakers, operational users, or even agent experts to understand and quantify workload for analysing the complexity of new case studies.

## Intended Readership

This document uses standard terminology to maximise the potential reach of readership, however some background and understanding of computer science is expected from the reader.

Diagrams were used to simplify the content surrounding complex structures and behaviours. These diagrams have been developed using Unified Modelling Language™ (UML) notation (see [www.uml.org](http://www.uml.org)). There is also an extension of UML called AUML, specifically for agent-based models, but it was avoided for simplicity.  
  
Instead, class elements were used to describe agent structure and interaction and activity diagrams were used to describe agent behaviours. Flow diagrams to describe details of algorithms and examples of coding are also included when needed.

Database diagrams use MySQL Workbench specific notation, instead of E/R (entity/relationship) notation. As reference, the differences between notations are:

* Cardinality of relationship uses standard UML style.
* Key fields have a key sign, foreign keys have a red rhombus sign and other fields have a blue rhombus sign.

## Structure

The first section describes the methodology, context and scope of the simulation software. The following sections document the enhancements; grouped into the following seven categories:

* Software architecture extension presents the updated software architecture and the changes to allow cloud deployment.
* Simulation dataset describes the changes in input data and the auxiliary scripts used to prepare initial dataset by pre-calculating data.
* Dynamic cost index introduces the optimisation process and the cost involved.
* Delay enumerates the delays applied to each flight phase
* Arrival and departure management covers both new processes
* Verification presents how the simulation is validated.
* Deployment provides a list of requirements and an updated guide to deploy the software.

## Methodology

There are several prominent methodologies for Agent Oriented Programming (AOP): Gaia (M. Wooldridge et.al., 2000.), MESSAGE (G. Caire  et.al., 2002), Prometheus (L.Padgham et.al., 2002), Tropos (F. Giunchiglia et.al., 2002) , Cassiopeia (A. Collinot et.al, 1996) and MaSE (F. Mark, A and Wood Scott, 2001). However, as some literature shows (Caire, G et.al., 2002), most methodologies focus on adapting object-oriented methodologies to agent based design and implementation.

In CASSIOPEIA, Innaxis contributed by providing a practical ad-hoc methodology for future agent-based simulation software in ATM. Due to the expertise gained in ABM and successful methodology from CASSIOPEIA, the team decided to follow the same line of work in the DCI-4HD2D project extension.

The first stage of the software development methodology is focused on the **selection and research of available agent-based simulation platforms and frameworks**. Then, the software architect assists the domain-specific ATM experts to **produce an agent based model** ready to run simulation with the basic components. Thereafter, the software architect starts to code different pieces of the model until each detail of the case study has been completed. ATM experts provide several validating scenarios to be implemented and select a set of output values from the simulations to validate the implementation. Once the model is both verified and validated, the ATM experts define the output dataset and the final set of scenarios parameters to be implemented.

Once the final case studies are implemented, the simulation tool is released and is ready to be used by ATM experts.

## Context

This agent framework extension is framed by the DCI-4HD2D project goal: **assessing the use of Dynamic Cost Indexing (DCI) in the context of the four hour door-to-door (4HD2D) objective via Collaborative Decision Making (CDM) procedures.**

## Acronyms and Terminology

| **Term** | **Definition** |
| --- | --- |
| ABM | Agent-based Modelling/Model |
| ABS | Agent-based Simulation |
| ADF | Agent Definition File |
| AMAN | Arrival Manager |
| AOBT | Actual Off-Block time |
| AOP | Agent-Oriented Programming |
| API | Application Programming Interface |
| ATM | Air Traffic Management |
| AUML | Agent Unified Modelling Language |
| BADA | Base of Aircraft Data |
| BDI | Belief Desire Intention model |
| CDM | Collaborative Decision Making |
| CSV | Comma Separated Values |
| DCI | Dynamic Cost Indexing |
| DMAN | Departure Manager |
| E/R | Entity Relationship |
| EOBT | Estimated Off-Block time |
| FL | Flight level |
| JADE | Agent middleware platform/framework |
| JADEX | A BDI-Agent System Combining Middleware and Reasoning |
| KPI | Key Performance Indicator |
| MAS | Multi-Agent System |
| NOP | Network Operating Plan |
| OOP | Object-Oriented Programming |
| SESAR | Single European Sky ATM Research Programme |
| SOBT | Scheduled Off-Block time |
| SQL | Structured Query Language |
| TD | Total delay for the flight |
| WFP | Wait for passengers |
| XML | Extensible Markup Language |

# Software Architecture Extension

The software architecture has been updated with a new layer with task automation components (previously, starting the simulation and acquiring results were automated through the Jadex Control Center and MySQL tools).

The architectural model Figure 2.1 shows the updated layered software architecture: new components are displayed in white and standard components inherited are grey.

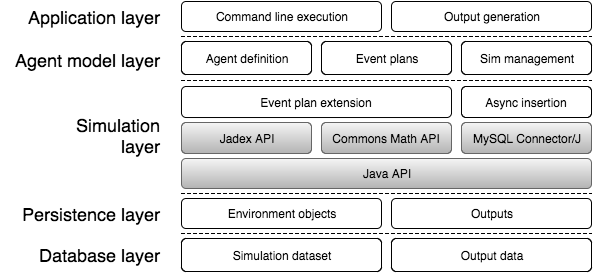


Figure . CASSIOPEIA simulation software architecture

The architecture consists of five layers:

* **Application layer** did not exist in previous case studies. Contains several components written in java for command line execution and generation of results into CSV files.
* **Agent model layer** has not changed from previous case studies. Contains required components to simulate the case, such as agent descriptions (in XML), agent behaviours (in Java) and simulation management.
* **Simulation layer** has not changed from previous case studies. Most of the components come from external libraries, but Jadex API and MySQL Connector/J are extended to enhance simulation clock and allow asynchronous insertion in database, respectively.
* **Persistence layer** contains some components which were defined on Database layer and Simulation layer in previous deliverables, such as Environmental objects and Output objects, written in Java.
* **Database layer** contains the same elements defined on persistence layer but focused on how they are organised in the database manager. They are written in SQL.

In addition to the changes on the software architecture, the simulation flow has also been updated because the use of external tools is no longer required to obtain results from simulations; the process is now fully automated. Figure 2.2 shows how the components work together to act as a simulator.

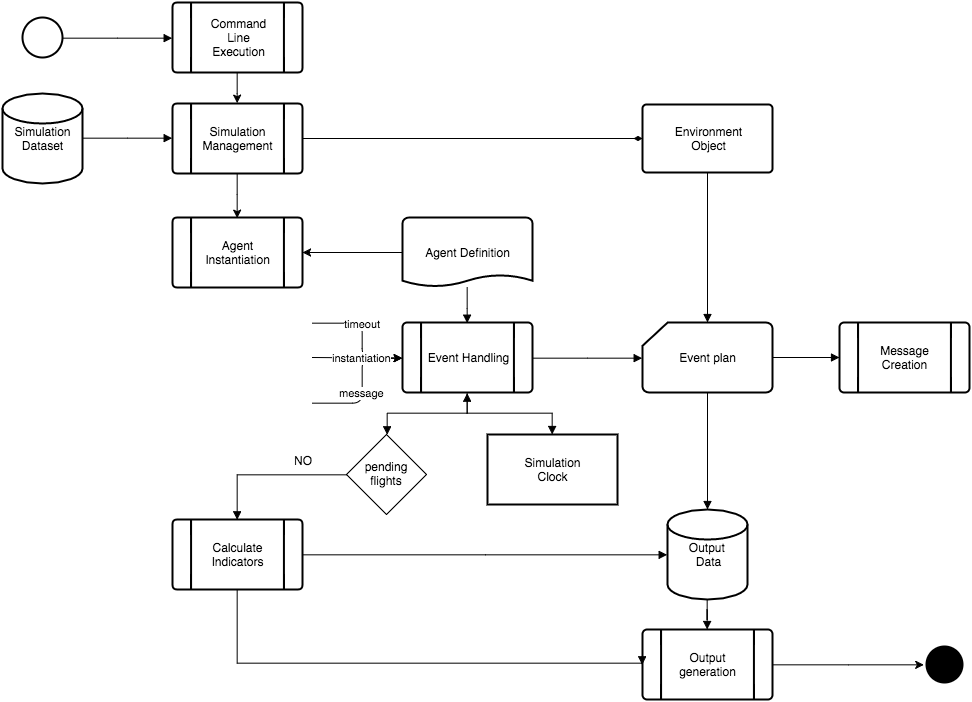


Figure . CASSIOPEIA simulation software workflow

By using the inputs provided by the user for the simulated scenario, a new simulation reference is created and the simulation dataset is obtained. Then, agents are instantiated according to the agent definition files. Each agent publishes an event once instantiation is done. The event handling process selects the event plan that corresponds with the published event. These event plans are context-aware and could modify the flight plans or send messages to other agents. When all flights have been simulated, simulation results are grouped and combined into indicators. All of the output data is then projected into result files that can be analysed by the user.

## Command Line Execution

Previously, the simulation required Jadex Control Center to be initiated. This restriction is not compatible in cloud environments, in which graphical user interfaces are not available. Jadex Platform API provides some methods to start a new simulation platform and the use of different Jadex services. The updated DCI-4H-D2D simulator is now a standalone application in which Jadex simulation platform has been encapsulated. Specifically, there are two different execution modes:

* **Define a new simulation scenario.**This execution mode creates a simulation scenario in the database before executing the simulation. This execution mode requires five arguments:
  + *-regulation\_name* The regulation name must have no spaces.
  + *-flight\_dataset* There are three predefined options, *0* (simulation dataset) ,*1* (ground improvements) and *2* (SESAR and ground improvements), consistent with number indicated in the database on flight table.
  + *-fuel\_cost*  The fuel cost fare must be a decimal number, ended with the letter 'f' (Float).
  + *-strategy*  There are three options, *FIXED*(no recover, for validation), *RECOVER\_DELAY* (baseline strategy) and *COST\_ORIENTED*.
  + *-initial\_delay*  There are three possible options, *NONE*, *LOW*, *MEDIUM* and *HIGH*.
  + -*compensations* There are three possible options, *NONE*, *CURRENT* and *FUTURE*.
* **Use an existing simulation scenario**. This execution model is useful for reproducing previous scenarios for validation, as well as for simulating the same scenario several times. This execution mode only requires one argument:
  + *-regulation\_id*  The value must be an integer value and must be defined into the database.

The simulator displays a warning message if arguments are not correct.

To illustrate, a few examples have been outlined. The first applies a new simulation scenario, called NOMINAL2; using the baseline dataset, 0.5 euros fuel cost, cost oriented strategy, low initial delay and no compensations. The last applies an existing simulation scenario identified by 1.

**Examples**

|  |  |
| --- | --- |
| 1  2  3  4  5 | # Execute a new simulation scenario  java -jar DCI4HD2DSim.jar -regulation\_name TEST -flight\_dataset 0 -fuel\_cost 0.5f -strategy COST\_ORIENTED -initial\_delay LOW -compensations NONE    # Execute an existing simulation scenario  java -jar DCI4HD2DSim.jar -regulation\_id 1 |

The simulator application performs the following activities on command line execution. Refer to Figure 2.3 for a graphical representation.

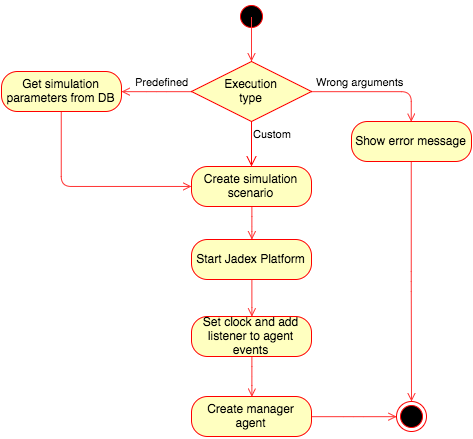


Figure . List of activities performed during simulation execution

First, the simulator checks application arguments. Predefined arguments should be taken from the database first, and incorrect arguments halts execution with errors. If there are no errors in creating the scenario, the simulation platform is first started and then configured. Platform configuration is focused on selecting required clock type, background platform execution, and handling of agent destruction events. Once the simulation platform is ready to use a simulation manager agent is instantiated. The execution control is then delegated to the manager agent until all simulation tasks have been completed.

## Automate Output

Previously, the results required MySQL tools to be exported into CSV files, the file format chosen by ATM specialists for validation and analysis, supported by data analysis tools such as Microsoft Excel. Again, this restriction is not compatible in cloud environments, in which graphical user interfaces are not available. With this update, CSV export is now performed inside the simulator application.

For example, the result files are stored in a separate folder. As the same simulation scenario could be simulated several times, each time a simulation is performed the results are located in a new folder using the current timestamp. If the current time is *2016/04/22 14:00:00*, the final result folder will be ***results/20160422140000/.***

The folder will contain several files. Each file has the same file name pattern: the first part identifies the regulation name and the last part identifies the type of file. There are four different file types, of which some could be hidden when performing predefined scenarios as they are usually generated for validation:

* *outputs:*This file contains the times simulated for each flight, estimated costs per phase, provided delays and selected speeds. This corresponds with view 'output'.
* *indicators:* This file contains system metrics calculated when the simulation has ended, such as delay, cost and speed variations. This corresponds with table 'indicators'.
* *proposals:* This file contains the proposed speeds and wait times (and related costs) when DCI is optimized. This corresponds with table 'proposals'.
* *costs:* This file contains final costs for flights, once the costs are allocated to the flights that generate the delay. This corresponds with view 'output costs'.

All files have the same format, CSV, which means Comma Separated Values. Some details of the format will depend on the current locale of the operative system, with the most compatible format used is the following:

* First row: HEADERS
* Field separation: Comma
* Decimal separator: Point
* Row separator: new line (MAC style)

There is an example of CSV costs output file located in Appendix A where the structure and format of the document is illustrated. The simulator application performs the following activities for generating these files types. Refer to Figure 2.4 for a graphical representation.

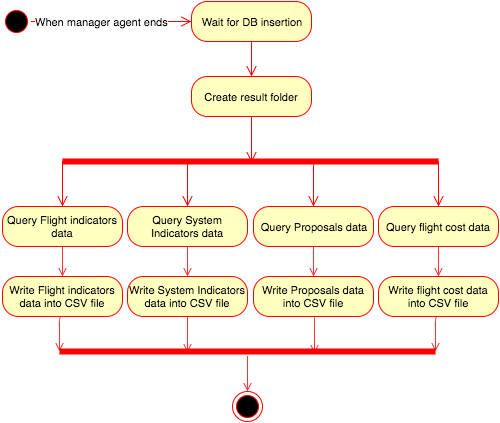


Figure . List of activities performed during CSV export

When the simulation manager agent ends all tasks, it notifies the platform that is going to "die". The simulation platform has designed to generate results immediately after the manager agent dies. The file generation starts by creating a results folder if it doesn't exist already. Before the results are taken from the database, the simulator checks if the results have already been stored into the database. Data is taken from database using a simple projection of all tables or view fields, depending on the number of tables required to perform the projection. Finally each field is written into the corresponding result file.

# Simulation Dataset

In particular, the simulation dataset has been updated in the following ways:

* Several aircraft models have been included that have additional available data for improving fuel estimations and flight speeds,
* Flight data have been extended to facilitate the analysis of input data.
* Simulation scenario data have also been extended to reproduce a wider range of situations.

From a database perspective, Figure 3.1shows the updated database model for the simulation dataset:

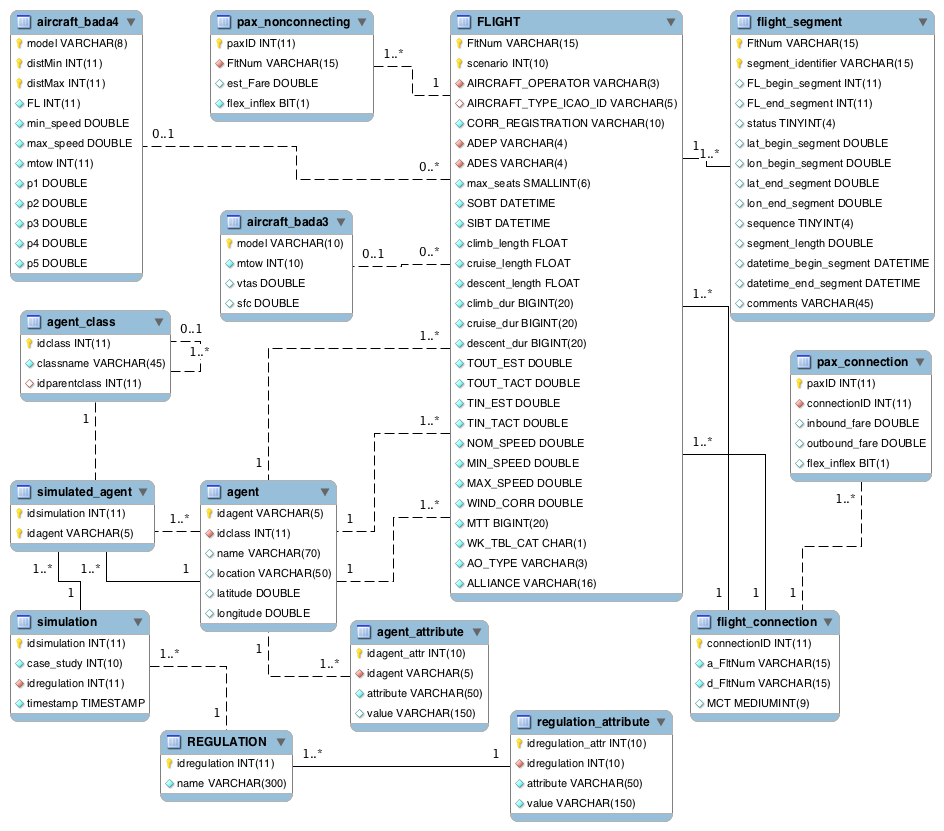


Figure . Simulation dataset database model

A new table called *aircraft bada4*, which is described later, along with two previous tables, *flight* and *aircraft bada3* have been modified. There are no changes in agent data or the simulation parameters, as the database approach allowed the necessary flexibility. Furthermore, the relations between data entities have not changed.

When the database layer changes, the persistence layer also changes. Figure 3.2shows the updated class model for the simulation dataset class model.

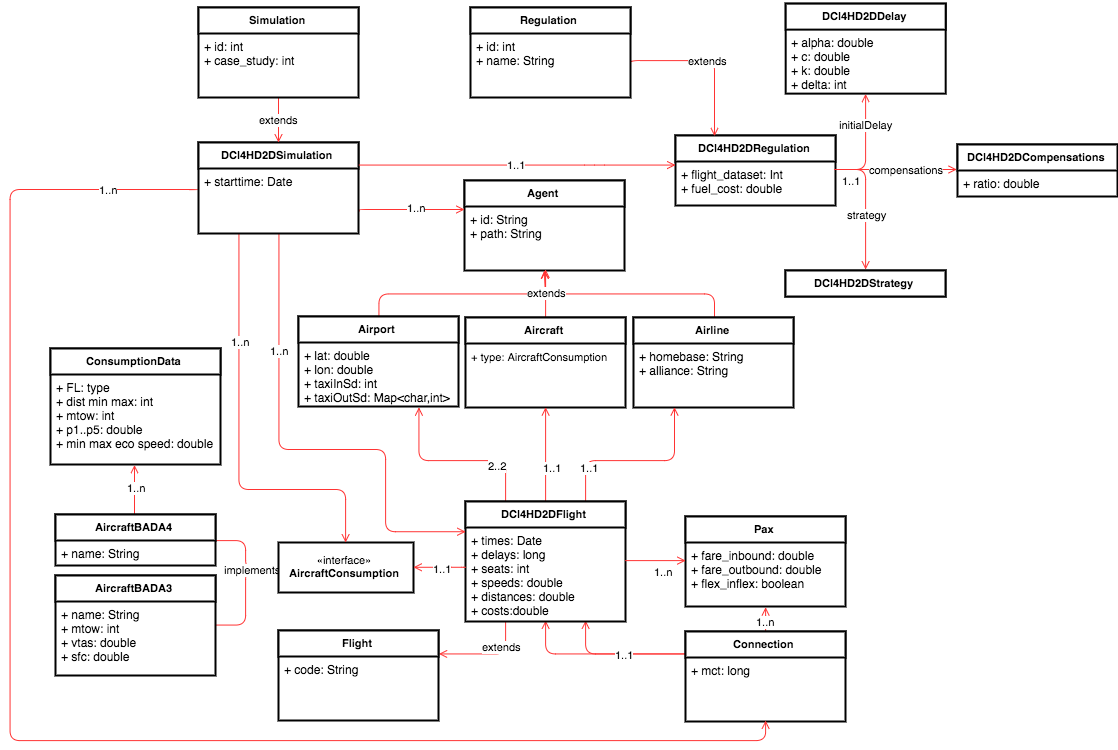
****

Figure . Simulation dataset class model

Other differences worth mentioning in the class model for the simulation environment include:

* Aircraft data classes have been re-written to maintain both versions of aircraft model data.
* Additional data for aircraft models depends on flight plan distance.
* New fields added in flight data, agent data and scenario parameters to fulfil simulation model data requirements.

More details regarding these changes are described in the following sections.

## Extend Flight Dataset with Pre-Computation

The previous dataset validation showed some inconsistencies in segment and flight data. Occasionally, those inconsistencies were too subtle to design a specific algorithm to fix them on an ad-hoc basis during the simulation. In order to address this issue, pre-computations have been performed. These pre-computed input datasets reduce the need of further computations during the execution of the simulations and facilitates the analysis of the validated input data.

For the same reason, the values in *ground improvements* and *SESAR ground improvements* tables to reproduce the expected improvements on flight plans were incorporated as additional flight scenarios. These tables have been removed once the migration of data was completed.

Many changes have been made to the flight table. These updates are described in Figure 3.3. In sum, these changes include:

* 18 fields have been added:
  + 5 flight phase durations fields (*taxi\_out\_est, climb\_dur, cruise\_dur, descent\_dur, taxi\_in\_est*), 3 distance-related fields (*climb\_dist, cruise\_dist, descent\_dist*), 2 tactical taxi times fields, 4 speeds fields and 3 additional fields for airline and aircraft parameters. There is one more additional field for scenario identification.
* 13 fields have been removed (e.g. due to redundancy or not needed):
  + 5 flight times (*IOBT, EOBT\_1, EOBT\_2, ARVT\_1, ARVT\_3*), 4 distances (*RTE\_LEN1, RTE\_LEN3*...), *flight consumption*, *cruising time* and the *exclude* flag.
* The link between *ground improvements* table and *flight* table has been broken, as *ground improvements* data is contained in another flight scenario.

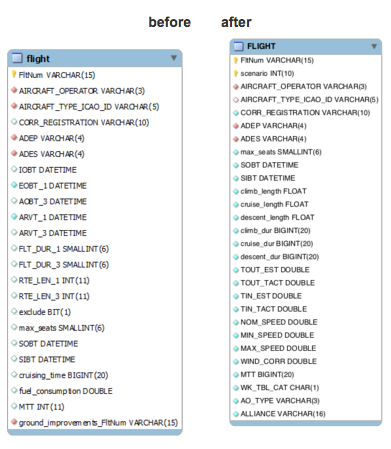


Figure . Flight table comparison

The algorithms that calculate the time intervals from segment data, such as cruise time, descent time and climb time, have been reviewed and re-factored into scripts that simply extend the original flight data. Ground and SESAR improvement algorithms have also been re-factored. Taxi times, both estimated and tactical, were added to the initial dataset manually.

Four scripts were coded to input the initial dataset in the simulator. Listed in order of execution, these scripts include:

1. **LOAD**. Script to load initial flight dataset from CSV in the database. It uses a MySQL-specific statement to load data once the developer defines the file path and the field mapping, so it should be executed in the database console.
2. **EXTEND**. Script to load flight times and distances for each phase from *flight segment* data. It is written in Java and some pieces of code were reused from an old version of the simulator code.

For each flight, the script first takes the flight segment data of a flight and calculates FLmax, which is the maximum FL for the flight (maximum of *FL begin segment* and *FL end segment* fields in *flight segment* table). Then, segment data, ordered by *sequence* field, is classified as follows:

* Segments which *FL begin segment* or *FL end segment* value is greater than FLmax – 40 are considered cruise segments
* Remaining segments are considered climb or descent segments, depending on their position in the list.

If resulting cruise or descent distances are greater than 200 NM, boundary climb or descent segments will be considered as cruise segments, to limit climb and descent distances to 200 NM.

1. **GROUND**. Script to get one flight dataset from database and apply ground improvements. It is written in Java.

For each flight, there is a 20% MTT reduction, minimum 20 minutes. If both connecting flights and previous aircraft flight have enough arrival buffers to apply MTT reduction, SOBT and SIBT are moved forward accordingly.

1. **SESAR**. Script to get each flight dataset from the database and apply SESAR improvements. It is written in Java

For each flight, orthogonal distance between airports is calculated and compared with scheduled distance. If orthogonal distance is less than scheduled distance, SOBT is delayed for inbound flights and SIBT is moved forward for outbound flights.

## Update BADA Datasets

The **aircraft model data** was also updated from the BADA 3 model to the BADA 4 model to improve fuel consumption estimations. There are a few aircraft models with no BADA 4 model available, and for these models BADA3 was used.

A new table called *aircraft bada4* is included in the database model. Instead of having one record for each *aircraft model type id*, there is now one record for each flight range, with fields for minimum, maximum speed, cruise flight level and MTOW, as well as 5 polynomial constants to calculate fuel consumption. These changes are described in Figure 3.4

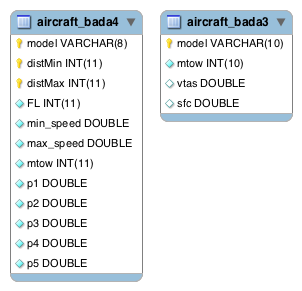


Figure . Structure of BADA tables

Aircraft data model is also updated from the persistence layer perspective. The previous aircraft model class is rewritten into *AircraftBADA3* class. The interface of the class was extracted into a common interface class called *AircraftConsumption*. New *AircraftBADA4* class implements this new interface and contains data for each flight range. Specific data for each flight range is contained in *ConsumptionData* class. These updates are described in Figure 3.5.

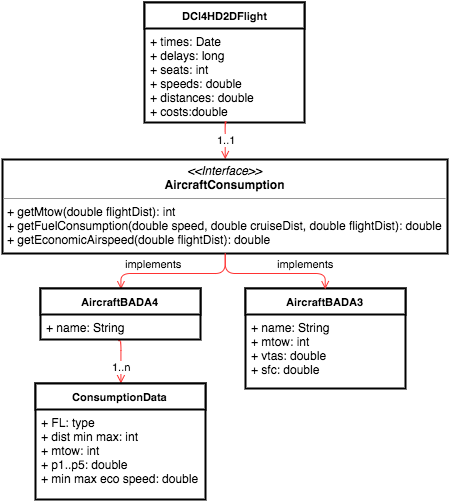


Figure . Aircraft data classes

The interface of aircraft data contains the following methods:

* *getMtow(double flightDist)* For BADA4 models, this depends on flight distance.
* *getFuelConsumption(double speed, double cruiseDist, double flightDist)* For BADA 4 models uses a 4 degree polynomial function and depends on the flight distance. For BADA3 models a quadratic function is used.
* *getEconomicAirspeed(double flightDist)*  For BADA 4 models, this depends on flight distance. Economic airspeed is calculated in advance when the simulation environment is created.

# Dynamic Cost Indexing

The main task that the airline agent performs is dynamic cost indexing.

The agent structure is described in figure 4.1, showing the list of attributes that define the agent state, as well as the tasks performed in response to agent communication. For this purpose, we are only focusing on the elements in bold, as the others are not used in dynamic cost index:

* *simulationDCI4HD2D* Contains the scenario parameters as well as flight connections
* *flights* Contains the list of flights for the airline.
* *optimizeFlight(requestOptimization)* Task performed in response to a *request optimization* message. It just performs the auxiliary operations required to perform DCI optimization, such as handling messages, save results and select the optimization algorithm.
* *getFlight*(flightid) Auxiliary method for finding out flights in the flight list by flight id.

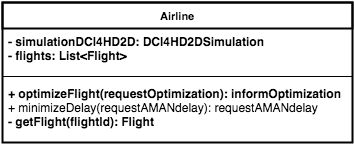


Figure . Airline agent structure

Dynamic Cost Indexing also uses some other classes, as shown in figure 4.2.

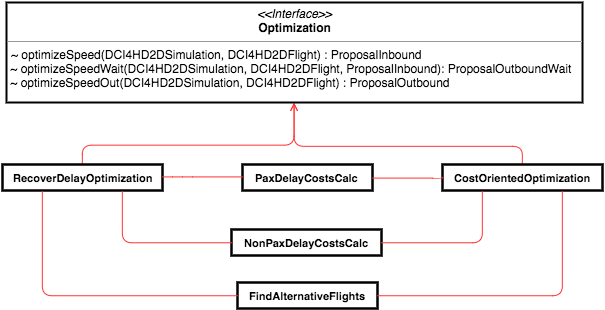


Figure . DCI optimization classes

Two classes, *RecoverDelayOptimization* and *CostOrientedOptimization*, implement the optimization methods, one for each DCI strategy. Another two classes, *PaxDelayCostCalc* and *NonPaxDelayCostsCalc*, contain methods to calculate passengers, crew and management delay costs. Finally, an additional class called *FindAlternativeFlights* implements the rescheduling of passengers and calculates rescheduling costs.

## DCI Optimisation

Dynamic Cost Indexing is applied at several points of each flight operation. The processes involved depend on the point in time and on the type of flight (inbound/outbound). Figure 4.3 shows the processes involved when DCI is applied for flights on the ground:

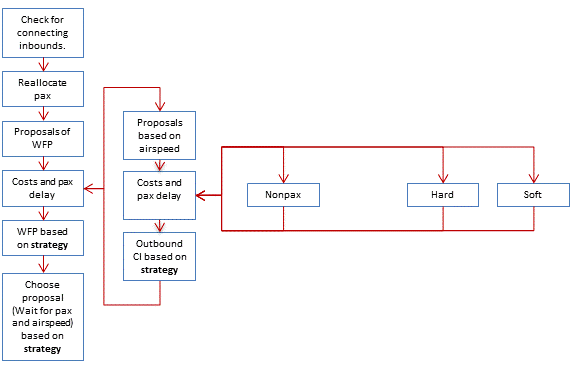


Figure . Outbound flight DCI processes on the ground

DCI is applied when flights are about to depart in outbound flights only, as inbound flights have no previous connecting data. DCI first takes the arrival delay for all inbound connecting flights and calculates which airspeed is required to recover the delay assumed to wait for those flights. The number of proposals will depend on the selected DCI strategy, but there will be at least one proposal per inbound flight that arrives before departing time.

Then, it calculates the delay for all passengers (connecting and non-connecting), crew and flight management activities. These delays will determine part of the costs of outbound delay, and the remainder of costs is related to airspeed and fuel costs. Rotational costs are estimated, as no connecting data is available at destination.

The proposal with lowest costs or arrival delay is selected; it will depend on the selected DCI strategy.

DCI is also applied when inbound and outbound flights reach top of climb. Figure 4.4 describes the processes involved in inbound flights when top of climb is reached:

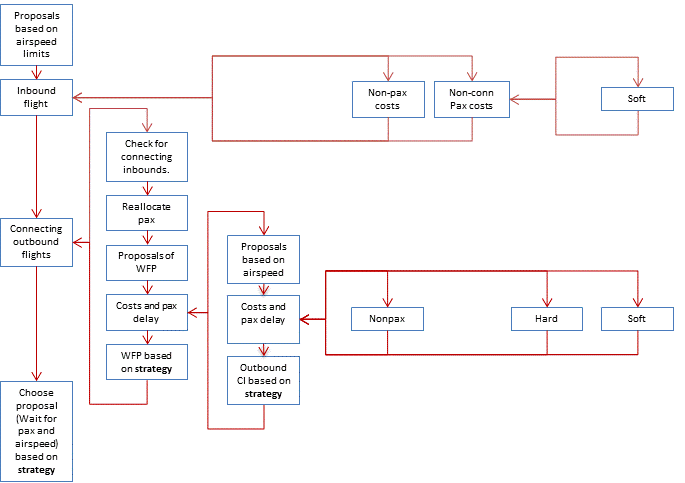


Figure . Inbound flight DCI processes on top of climb

Outbound flights apply DCI each time an inbound connecting flight changes its airspeed. So, each inbound flight DCI proposal considers the costs of outbound delay as well as the cost of inbound delay when minimizing delay costs. The costs of inbound delay will depend on the same delays enumerated for outbound flights at ground, except from connecting passenger delay cost that are not calculated to avoid double counting.

The number of inbound flight DCI proposals will depend on the airspeed range and the estimated delay at arrival. In this case, rotational cost is calculated using previous flight EIBT. Depending on the DCI strategy, the option with minimum delays of minimum costs will be selected.

Outbound flights also apply DCI when they reach top of climb, but in this case only outbound costs are considered. Figure 4.5 describes the processes involved:

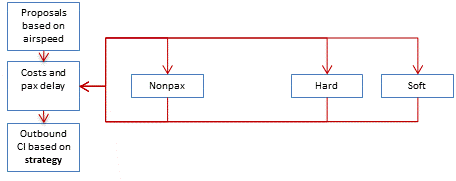


Figure . Outbound flight DCI processes on top of climb

Considering the computation times exceeded workable limits the first time the whole process was performed, several computing time optimisations were needed:

* **Disable cost assignation to airlines:** On one hand, it is hard to predict in advance which airline should pay the generated delay, as there could be more than one flight from different airlines that makes an outbound flight wait. On the other hand, we do not require that level of detail in order to determine the optimal option. In any case, costs are assigned to airlines when results are analysed, so disabling this computation during the simulation would not be a limitation. The internal algorithm for calculating costs is greatly simplified, as it does not require airline data and temporary variables to manage airline costs. Output validation fields for airline costs were also removed.
* **Reduce optimisation options**:For the previous approach, there are too many delay recovery options considered. The differences in airspeed between different recover delay proposals were 0.1 km/min. This value could be too small for short haul flights. Instead of changing airspeed between proposals, we are now considering a delay recovery. Each proposal recovers 1 additional minute of delay. Additionally, proposals when arrival delay has already been recovered are not calculated.
* **Reduce times when DCI optimisation is performed**. Instead of performing DCI each flight phase, DCI is performed at cruise and when deciding the wait time for passengers. The following table describes the times when DCI is performed and which costs are then calculated:

Table Cost estimation and flight phases

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Inbound aircraft | | | | | | | | | Outbound aircraft | | | | | | | | | | |
|  | Non-pax | | | | | | | pax | |  | Non-pax | | | | | | | pax | | |
|  | At gate (maint) | Taxi out (maint) | En route (maint) | arrival (maint) | Taxi in (maint) | crew | fuel | hard | soft | Wait for pax | At gate (maint) | Taxi out (maint) | En route (maint) | arrival (maint) | Taxi in (maint) | crew | fuel | hard | soft | Rotational |
| ARCT | final | final | y | 0 | 0 | y | y | y | y | y | y | 0 | y | 0 | 0 | y | y | y | y | y |
| AMAN | final | final | y | y | 0 | y | y | y | y | y | y | 0 | y | 0 | 0 | y | y | y | y | y |
| EOBT |  |  |  |  |  |  |  |  |  | final | y | 0 | y | 0 | 0 | y | y | final | y | y |
| ARCT |  |  |  |  |  |  |  |  |  |  | final | final | y | 0 | 0 | y | y | final | y | y |
| Analysis | final | Final | final | final | final | final | final | final | final |  | final | final | final | 0 | final | final | final | final | final | final |

After evaluating results from simulating several validating scenarios, analysis showed minor improvements in arrival delay and airline costs. Some features were added to improve optimization performance, including:

* **Dynamic rotational delay**

Outbound flights cannot depart until an inbound flight with same aircraft arrives at the airport and the aircraft is ready to serve the next flight. Previously, rotational delay was not estimated, so DCI does not consider rotational delay until outbound EOBT, when DCI is performed again. The rotational delay is also considered when an inbound selects a different DCI option, in which case the rotational delay is estimated using EIBT.

* **Cruise trajectory**

When the cost index of a flight is increased, not only is its cruise speed modified but the whole trajectory might be affected. However, DCI optimization is requested once the flight reaches TOC, therefore, changing flight level is not considered. Instead, cruise length is incremented and descending length is decreased the same amount of distance when the flight speeds up.

The amount of distance change is randomly calculated from a Normal distribution of parameters μ=7.60 minutes and σ=2.15 minutes. The amount of distance change is the same for every time the DCI optimization is performed. Once the evaluated speed is greater than nominal speed, time intervals are modified to simulate the effect of this distance change.

As the descent time could change as could the cruise time, DCI optimization algorithm has been updated to calculate recovered delay using the descent time as well. Normal distribution model takes part of the Apache commons-math3 library.

## DCI Strategies

When DCI optimisation is performed, the model considers two different strategies for recovering delay:

* The baseline approach, where delay costs have no impact on the selected DCI proposal,
* The minimum costs approach, where the selected DCI proposal only depends on delay costs

|  | **When is delay recovery triggered? (MIN\_DELAY)** | **What drives the minimum residual delay? (MAX\_RESIDUAL)** | **Wait for passenger rules (MAX\_WAITING)** | **To which flights does delay recovery apply?** |
| --- | --- | --- | --- | --- |
| **Baseline (RECOVER DELAY)** | Arrival delay > 15 minutes | Fixed at 5 minutes | Inbound flight recovering delay and waiting time required < 20 minutes | 10% of flights |
| **Minimum cost (COST ORIENTED)** | Arrival delay > 0 minutes | Limited by total cost efficiency | Driven by total cost efficiency and waiting time < 60 minutes | 100% |

The DCI strategy is specified in the simulation scenario, as indicated by the user before the simulation. Each strategy is declared in an Enum type.

Reviewing figure 4.6:

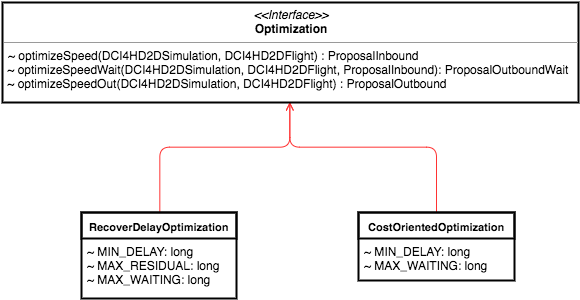


Figure . Class structure for DCI strategies

There are two classes that implement the optimisation interface: *RecoverDelayOptimization* class for the baseline strategy and *CostOrientedOptimization* class for the minimum cost strategy. Both have several attributes, such as MIN\_DELAY, MAX\_RESIDUAL and MAX\_WAITING, to create different options for recovering delay. Each class also has three methods:

* *optimizeSpeed(input dataset, inboundFlight)* Recovers delay for inbound flights when reaching top of climb. It depends on the OptimizeSpeedWait method for outbound costs calculation.
* *optimizeSpeedWait(input dataset, outboundFlight, inboundFlight, inboundProposal)*  Generates the smallest amount of delay by waiting connecting flight and recovers the biggest amount of departure delay for outbound flights when they are about to depart.
* *optimizeSpeedOut(input dataset, outboundFlight)*Recovers delay for outbound flights when reaching top of climb.

There are some differences in DCI optimisation algorithm between the two strategies:

* **Baseline strategy** calculates one optimal recovery option only if the flight is delayed using nominal DCI and it is one of the 10% of flights that could recover delay. The optimal option would have the maximum speed available so that it provides an arrival delay greater than residual delay. **Baseline strategy** always waits for an inbound if it is speeding up and required departure delay is less than 20 minutes.
* **Minimum cost strategy**calculates each DCI option from nominal speed to max speed always. The costs for each option will be sorted and the option with lower costs will be selected.   
  Outbound flights checks before optimization if any connecting passenger will miss the connection with current schedule. Waiting time is calculated for each group of passengers from the same inbound flight. The list of options will be a permutation of the different waiting time and speed DCI options**.**The costs will be sorted and the option with lower costs will be selected.

## DCI Costs

The following costs are also considered when evaluating different DCI proposals:

**Passenger Delay Costs**

Passenger regulations have evolved since the introduction of Regulation 261 in 2005 [[1]](#footnote-1)[1]. According to Regulation 261 and subsequent amendments [[2]](#footnote-2)[1], passengers are entitled to compensation when their flight is delayed on arrival. The following table defines the compensation each passenger is entitled to:

| **Flight distance** | **Arrival delay** | **Compensation** |
| --- | --- | --- |
| 1,500 km > d <= 3,500 km | >=180 minutes | €400 |
| <=1,500 km | >= 180 minutes | €250 |
| > 3,500 km | 180 minutes < r <= 240 minutes | €300 |
| > 3,500 km | > 240 minutes | €600 |

Not all passengers seek compensation, e.g. due to lack of awareness of their entitlement. We've estimated that 11% (ref) of passengers currently apply for such compensation. We can also increase this estimation up to 50% to evaluate if airlines would change DCI options when passengers were more prone to apply for compensation. There are other options for when Regulation 261 is not applied:

* No compensation ratio (for validation only): 0 % passengers.
* Current compensation ratio: 11 % passengers.
* Future compensation ratio: 50 % passengers.

Those costs are modelled into *PaxDelayCostCalc* class in a new method called *hardCostsCompensations*. The amount of costs will depend on the amount of delay, number of passengers and selected compensation scenario described in figure 4.7, modelled on *DCI4HD2DCompensation* object.

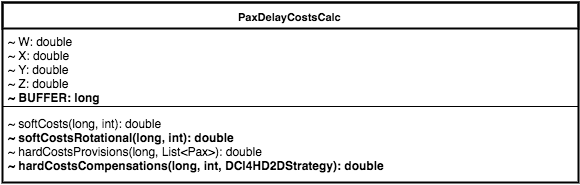


Figure . Passenger costs class model

Furthermore, as we cannot calculate rotational delay for flights that depart from airports other than Zurich because input model has only connection data from Zurich, we defined a new method called *softCostsRotational* to estimate those costs. We consider that each connection has an arrival buffer (BUFFER attribute) of 30 minutes and flights with larger arrival delay should have additional delay soft cost in flight passengers, because they could miss the next connection. So, if a flight has 75 minutes of delay, the next two connections could be missed (which is the maximum number of next missed connections), therefore additional soft cost of 45 minutes and 15 minutes are considered. Those costs will depend on the amount of delay and the number of passengers.

**Non-Passenger Costs**

Non-passenger delay costs have also changed. Management cost fares are cached for each flight. Figure 4.8 describes the updated *NonPaxDelayCostsCalc* class.

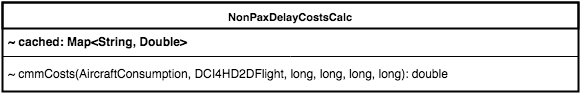


Figure . Non-passenger cost class model

# Delay

The first iteration of the CASS model simulated the scenarios with no-delay uncertainty, as the system needs to be predictable in order to be validated. Once the simulator was validated sufficiently, delay uncertainty was added to evaluate the adaptability of the proposed DCI model.

We consider three different types of delay:

* **Off-Block delay:** The amount of delay depends on *input delay* scenario parameter. The estimation of off-block delay will change as time goes by until actual off-block time. Delay is calculated taking one random value from a Burr Distribution of parameters defined as attributes of *DCI4HD2DDelay* enum type. That distribution was not included in Apache commons math3 library, so a custom implementation was developed.
* **Taxi delay uncertainties:**Tactical taxi times were affected by some uncertainty. The method *taxiTimeUncertainty* in *FlightSimulationPlan* class takes one random value from a Normal probability distribution, where the mean of the distribution is the tactical time and the standard deviation is taken from the input dataset, and returns as the actual taxi delay. Standard deviation taxi times from input dataset are modelled on Airport class, as the value depends on the airport. This class has two methods, *GetTaxiInSD* and *GetTaxiOutSD* to get standard deviation of taxi in time and taxi out time respectively.
* **Airborne delay**: The current simulation model considers two different delay uncertainties for airborne delay: climb and cruise uncertainty. The value is taken randomly from a Normal distribution of parameters defined as constants in the methods *ClimbUncertainty* and *CruiseUncertainty* methods in *FlightSimulationPlan* class.

## In Block Delay

In Block delay is calculated when flights are instantiated. Then, depending on the time the delay is required, there will be a different amount on delay, as seen on figure 5.1.



Figure . Initial delay generation for flights

At TEOBT1 the airline is notified that the EOBT is EOBT1.At TEOBT2 the airline is notified that the EOBT is EOBT2.At AOBT, the airline is notified that the AOBT is AOBT. For inbound flights, only AOBT is considered. The following steps are performed when calculating In Block delay:

* For inbound flights:
  1. Generate d=rand(0,1)
  2. Generate total delay for the flight TD=icdtd(d)
  3. EOBT = SOBT + TD
  4. AOBT = EOBT
* For outbound flights:
  + Two delays will be computed: the actual delay assigned to the flight TD and a second delay to generate the first estimation of the airline off block time EOBT1. This second delay will be computed in the cumulative probability range of the first one:  icdtd(d±rand(-0.2,0.2)).

1. Generate d=rand(0,1)
2. Generate total delay for the flight TD=icdtd(d)
3. AOBT = SOBT + TD
4. Generate final uncertainty: FU = rand(normal(μ=0,σ=3)
5. EOBT2 = AOBT + FU
6. EOBT1 = SOBT + icdtd(d±rand(-0.2,0.2))
7. Generate TEOBT1 in the range [SOBT-4h00, min(EOBT1-0h15, SOBT-0h15)]
8. Generate TEOBT2 in the range [min(EOBT1-1h00, EOBT2-1h00), min(EOBT1,AOBT)]

A class in the persistence model, called DCI4HD2DDelay declares the available values for *input delay* scenario parameter, as well as the constant value for each parameter of the Burr distribution and the method to calculate the inverse of the cumulative distribution.

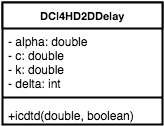


Figure . Delay enum type

## Taxi Uncertainty Values

The flight dataset has estimated and tactical data for taxi times. The actual taxi time is calculated using a normal distribution, and by using tactical taxi time as the mean and the standard deviation value. Standard deviation values were then uploaded in the database using *agent attribute* table, described in figure 5.3:

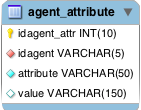


Figure . Agent attribute database model

Taxi standard deviation times are saved in the database using the ICAO name of the airport as *idagent* and one of the following attributes:

* For taxi-out: TXOSD\_"WK\_TBL\_CAT". Example: TXOSD\_M
* For taxi-in: TXISD

Airport data objects were extended with two new attributes, called *taxiInSd* and *taxiOutSd*. The last item is a map object in which its key is the *wk\_tbl\_cat* type. Figure 5.4 describes airport data structure.

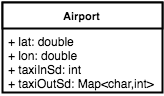


Figure . Airport persistence model

Specifically, the taxi time should be greater than 2 minutes and less than tactical taxi time plus two times the standard deviation. If it is not, taxi time is then calculated again.

## Airborne Uncertainties

There are two other delays that are considered in airborne: climb time and cruise distance. Both are taken from a normal distribution with different parameters:

* **Climb time:**mean -0.4,SD 4.3 minutes.
* **Cruise distance:**mean-12.2, SD18.7 kilometers.

Both delays are calculated in FlightSimulationPlan agent behaviour as auxiliary methods.

# Arrival and Departure Management

We model the arrival and departure manager to not exceed the capacity of an airport using DCI optimisation. We reused the airport agent for this purpose, as previous airport responsibilities, such as notifying off-block and in-block delay, are also covered in arrival and departure management. The new agent structure is described in figure 6.1:

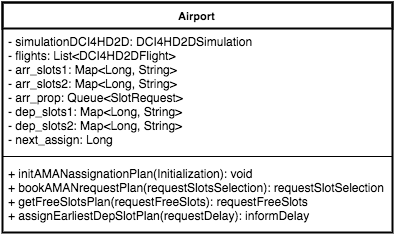


Figure . Airport agent structure

This agent has the following list of attributes that define the agent state:

* *simulationDCI4HD2D* Simulation scenario
* *flights* List of flights in the airport
* *arr slot1* and *arr slot2* Assignment of arrival flights to slots (2 flights each 3 minutes)
* *arr prop* Slot requests pending to assign
* *dep slot1* and *dep slot2* Assignment of departure flights to slots (two flights each 3 minutes)
* *next assign* Time for next round slot assignment.

There is only one task required for departure management called *assignEarliestSlotPlan*. This task is performed in response of a request delay message. DMAN delay will depend on the simulation scenario, available slots in 'dep\_slot1' and 'dep\_slot2' and flight parameters, such as flight speed.

There are three additional tasks for arrival management called *initAMANassignationPlan*, *getFreeSlotsPlan* and *bookAMANrequestPlan*. The first one, *initAMANassignationPlan*, is performed when the agent is instantiated, but it also assigns arrival slots every 5 minutes. AMAN delay will depend on the simulation scenario, received slot requests, flight parameters, available slots in 'arr\_slot1' and 'arr\_slot2' and the time left until the next round for slot assignment.

## DMAN: FIFO Approach

Outbound flights ask for DMAN delay after the airline of the flight decides if the flight needs to wait for passengers or not. The DMAN approach is described in figure Figure 6.2:

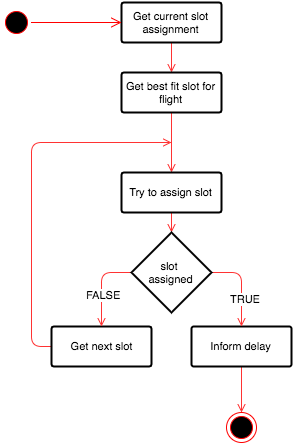


Figure . Calculate DMAN delay approach

The algorithm finds the slot that fits better for the departure flight. It tries to assign that slot, and if successful, it informs the DMAN delay through an inform message. If unsuccessful, it attempts the next slot and keeps trying until there is a slot for the flight available.

## AMAN: Arrival by Priority

AMAN negotiation is more complex than DMAN negotiation, as shown in figure 6.3. Three agents are involved, as the airline acts as a middleman to ensure that the selected slot minimises delay.

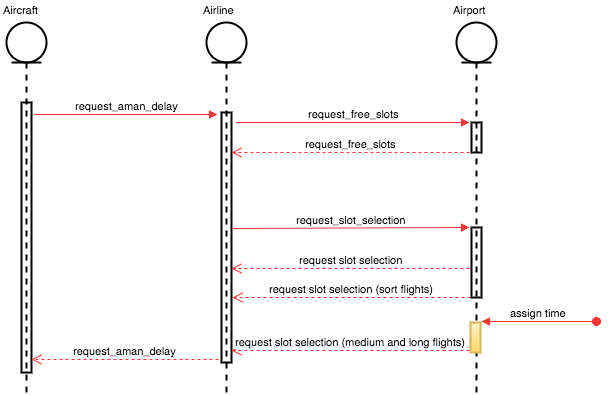


Figure . AMAN delay negotiation

**Aircraft - Approximation task**

Sends a message to the airline to start AMAN negotiation and wait for response, and then modifies times because of DMAN delay.

**Airline - Minimize delay task**

Prioritises free slots by minimal cost and sends the preference back to the airport as well as sending and receiving messages.

**Airport - Get free slots task.**

Starting with the best slot for max speed, the system checks if it is free. If not, system tries next slot until the best slot is found for eco speed was checked. If no slots are free, it continually looks for a free slot.

**Airport - Book AMAN request task**

*For short flights:*Perform assignment.

*For medium and long flights*: Just mark the proposal as pending.

**Airport - Init AMAN assignation task**

For every 5 minutes, the airport assigns a slot for pending requests by priority. If none of the options are free, it continues to look for a free slot to ensure each flight has an arrival slot.

# Validation & Analysis

The validation checks the accuracy of the proposed model. Considering traditional validations compare results from real systems or previous studies, a traditional validation could not be performed since such data for a proper comparison does not exist. Therefore, the approach taken is validating the main features of the model one by one, such as:

* Recovering delay and wait for passengers
* Flight costs
* Estimations of delay and other calculations

For the analysis, the emergent behaviour was examined from results. The validation of the model provides a thorough understanding of how the model behaves which is used in the design of metrics, which in turn allows a comparison of how the model behaves in different scenarios.

Generated results in both tasks are saved into the database. Figure 7.1 describes the scheme of the output data database model:

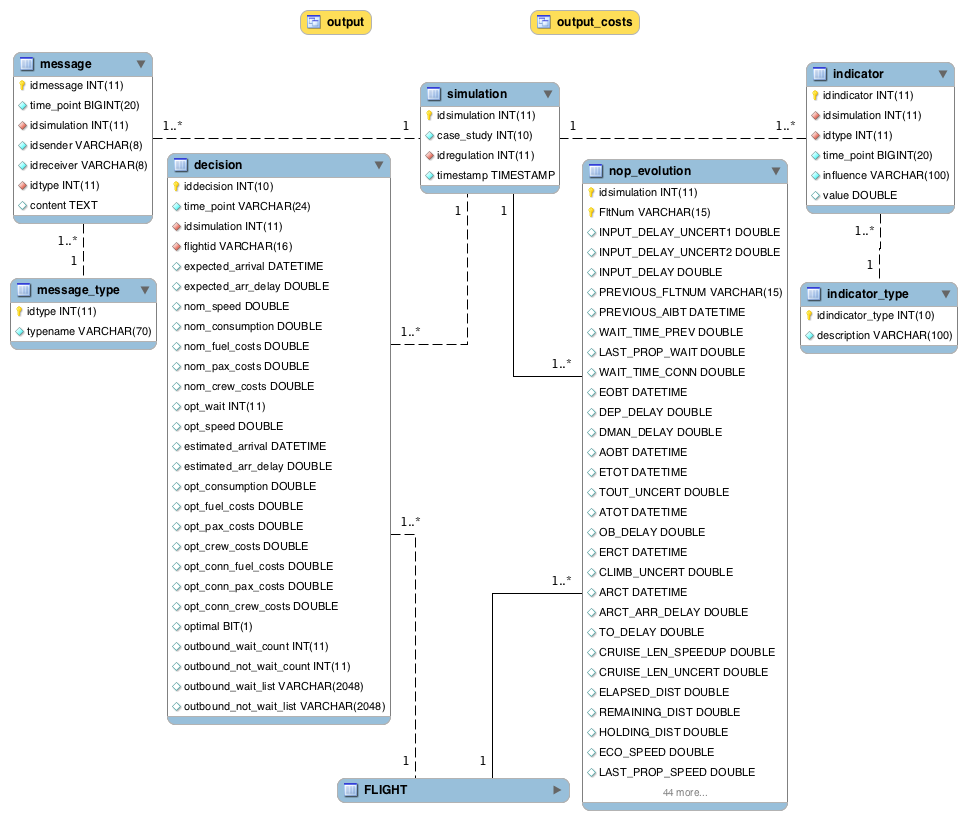


Figure . Output database model

The table structure is similar as previous case studies, however most of the fields have been changed in tables *decision* and *nop evolution*, in order to reproduce the required fields for validating.

Similar to the database model, *Decision*and *DCI4HD2DFlight*classes have been changed, which correspond with *decision* and *nop evolution* tables.

Figure 7.2 displays the hierarchy between classes. Note that DCI4HD2DFlight attributes and methods are not shown.

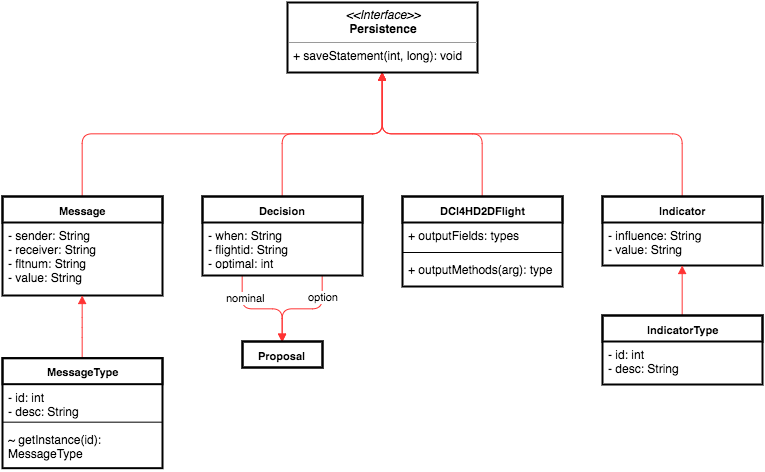


Figure . Output persistence model

## Validate DCI Speed Selection

The intermediate results from DCI optimization are modelled in the Proposal classes. Common attributes are located in the Proposal class and specific attributes are located in *ProposalInbound*(for inbounds), *ProposalOutboundWait*(for outbound flights on the ground) and *ProposalOutbound*(for outbound flights out of the ground). Figure 7.3 shows the hierarchy between proposal classes.

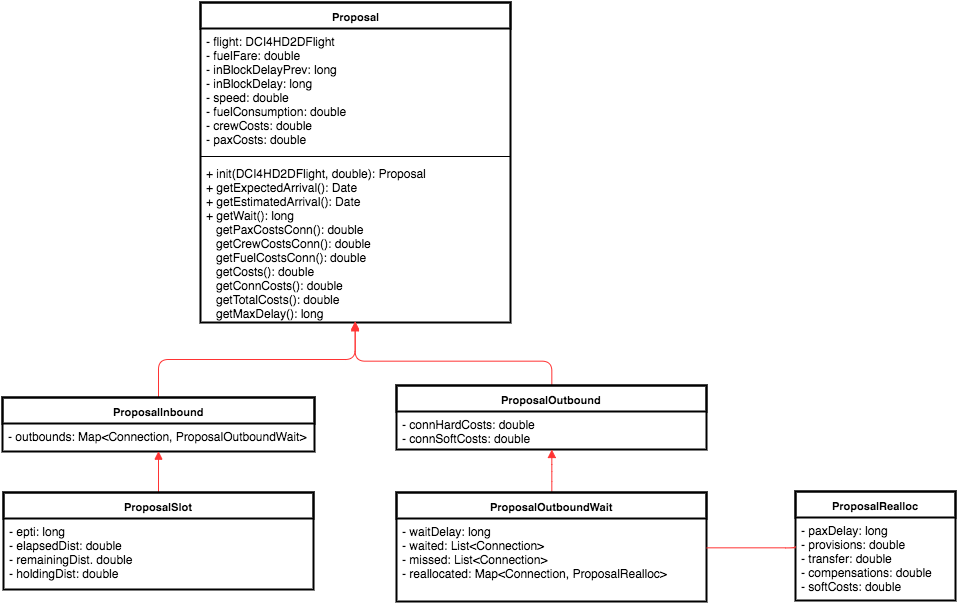


Figure . Hierarchical model of proposal classes

The proposals contain the expected delay and related costs for one of the possible speeds. The proposal data is saved in the database in the decision table for validating the process. The decision table also contains data from nominal proposal for reference. Figures 7.4 and 7.5 describe decision structure both from the persistence and database perspective.

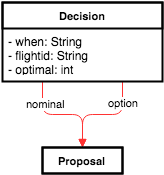


Figure . Decision class in persistence model

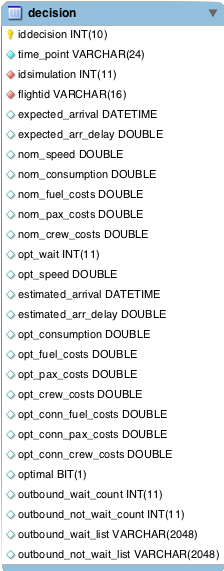


Figure . Decision table in database model

The following reflects the mapping between decision and proposal fields:

* *when* -> Four possible values: INBOUND\_ARCT, INBOUND\_AMAN, OUTBOUND\_EOBT2 and OUTBOUND\_ARCT
* flight\_id -> option.flight.getCode()
* expected\_arr\_delay -> nominal.inBlockDelay
* nom\_ fields -> nominal. fields
* estimated\_arr\_delay -> option.inBlockDelay
* opt\_ fields -> option. fields
* optimal -> YES or NO (Boolean)
* outbound\_ fields -> generated dynamically in code

## Validate Flight Simulation

The intermediate times, delays and costs are shown in a database view called *output*. More than 100 fields are calculated and recorded during the simulation of the flights. Some fields include:

* Selected speed at certain times and related costs
* Flight delays for each flight phase
* Final times for each flight phase
* Flight parameters for reference

Actual delays and costs are calculated when creating indicators and are shown in a separate view, called *output costs*:

* *Idsimulation* Simulation identifier
* *FltNum* Flight code
* *AO\_TYPE* Type of airline
* *AIRCRAFT\_TYPE\_ICAO\_ID* Type of aircraft
* *ARR\_DELAY* Delay at arrival regarding SOBT
* *ANHC* Actual non connecting hard costs
* *ACHC* Actual connecting hard costs
* *ARHC* Actual rescheduling hard costs
* *APSC* Actual passenger soft costs
* *AFC* Actual fuel costs
* *ANPC* Actual non passenger costs

In both cases, values in bold are saved first in *DCI4HD2DFlight* and then inserted into the database in *nop evolution* table, using *FltNum* key. The remaining values are taken from *flight* table.

## Scenario Analysis

Several flight characteristics are selected to compare between simulation scenarios. These attributes are summarised using Statistical libraries and added to the database to be analysed by ATM experts. Figure 7.6 describes the process.

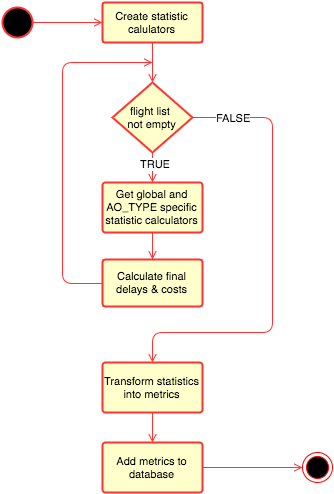


Figure . List of activities in scenario analysis

The flight attributes considered in metrics calculation are:

* Departure delay => Provision costs
* Connecting delay (for connecting passengers) => Provision costs
* Arrival delay (for connecting passengers, outbound arrival delay) => Soft cost and compensation costs
* Holding time (AMAN)
* Speed variation => Fuel costs
* Wait for passengers time
* Gate to gate passenger time
* Missed connections
* Crew & management costs

The algorithm uses several functions taken from an external library called Apache Commons Math3. Figure 7.7 describes when those functions are used and what the signature of the function is, as well as the class name.

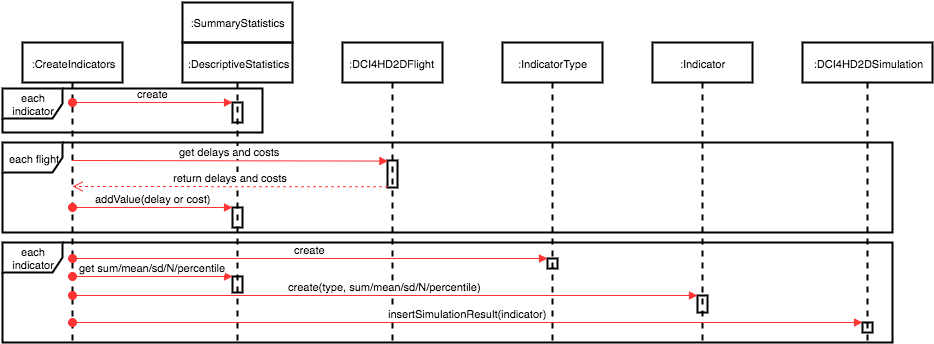


Figure . Interactions between analysis classes

# Deployment

Prior to the deployment of the simulation environment, there were several iterations in which relevant changes were gradually implemented. Such iterations included substituting specific methods from required external libraries to standard libraries and removing Jadex configuration files.

The following are some basic requirements and an updated step-by-step guide for deployment:

## Requirements

* The database server must be deployed in the same instance where the simulation tool is executed. It is useful to review *DBAsyncAdapter* class if you the database cannot be deployed in the same instance.
* There is only one thread simultaneously for simulating behaviours, so increasing CPU cores will not improve performance.
* At least 1GB of free memory is recommended before start the simulation

## Step-by-step guide.

The guide was tested on OSX (Mac) and Windows desktop systems, as well as Amazon Linux cloud system.

1. The simulation tool will be delivered in JAR format and external libraries will be packaged into. The JAR file can also be generated from source code using standard JAR runnable export from Eclipse using default settings (check that eclipse consider XML files as source code also).
2. Database dump file will be also delivered with database schema and input dataset. It was tested in MySQL Community Edition 5.5 and 5.6
3. Install MySQL server 5.5
4. Execute dump file into the database server.
5. Grant full access for user *cass* and password *cassiopeia* in database *cassiopeia\_cs3*
6. Install JRE 1.6 at least.
7. Generate script file if necessary to simulate several scenarios autonomously.
8. Create results directory
9. Execute the script on background and ignoring hangup signal before disconnecting ssh terminal. See nohup man page.
10. Nohup will redirect standard output to nohup.out file. It will contain simulation log.
11. Results files are saved in results folder

# Appendix A – CSV Cost Output File

The provided example was extracted from a CSV output cost file generated by simulating one of the testing scenarios. It contains the first 20 lines of the file.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | idsimulation,FltNum,AO\_TYPE,AIRCRAFT\_TYPE\_ICAO\_ID,ARR\_DELAY,ANHC,ACHC,ARHC,APSC,AFC,ANPC  1766,AAL\_KJFKLSZH01,FSC,B763,-3.98,0,0,0,0,18140.63,415.28  1766,AAL\_LSZHKJFK01,FSC,B763,-20.23,0,null,null,0,35601.54,32.31  1766,ACA\_CYYZLSZH01,FSC,B763,54.62,0,0,83,592.49,19184.79,1270.65  1766,ACA\_LSZHCYYZ01,FSC,B763,13.39,0,null,null,20.84,37128.55,195.68  1766,ADR\_LJLJLSZH01,FSC,CRJ9,48.12,0,0,0,124.58,157.18,408.63  1766,ADR\_LJLJLSZH02,FSC,CRJ9,4.85,0,0,0,0.87,114.52,31.77  1766,ADR\_LSZHLJLJ01,FSC,CRJ9,16.82,0,null,null,8.47,334.2,236.53  1766,ADR\_LSZHLJLJ02,FSC,CRJ9,-0.51,0,null,null,0,191.66,8.05  1766,AFL\_LSZHUUEE01,FSC,A319,-5.78,0,null,null,0,6682.57,5.31  1766,AFL\_UUEELSZH01,FSC,A319,-0.52,0,0,0,0,3294.93,109.67  1766,AMC\_LMMLLSZH01,FSC,A319,-6.58,0,0,0,0,1606.43,58.76  1766,AMC\_LSZHLMML01,FSC,A319,8.24,0,null,null,4.03,1787.4,137.79  1766,AUA\_LOWWLSZH01,FSC,B737,53.79,0,0,166,220.5,354.23,483.33  1766,AUA\_LOWWLSZH02,FSC,B736,12.33,0,0,83,4.41,256.05,220.08  1766,AUA\_LOWWLSZH03,FSC,B738,-24.95,0,0,0,0,263.28,0  1766,AUA\_LOWWLSZH04,FSC,A320,-4.11,0,0,0,0,268.24,0  1766,AUA\_LSZHLOWW01,FSC,A320,11.57,0,null,null,9.95,1028.93,139.59  1766,AUA\_LSZHLOWW02,FSC,B737,26.75,0,null,null,85.36,741.1,187.56  1766,AUA\_LSZHLOWW03,FSC,B736,-21.87,0,null,null,0,1202.04,10.42 |

The first line enumerates the fields contained in file, in this case, the ones required to validate final costs according to final arrival delay. Values are separated by commas and all fields in a row correspond with values from a flight.

1. [1] The cost of passenger delay to airlines in Europe – University of Westminster [↑](#footnote-ref-1)
2. [↑](#footnote-ref-2)