|  |  |
| --- | --- |
| SESAR_JU | |
| E.02.14 - DCI-4HD2D Platform extension report V1 | |
| Document information | |
| Project Title | DCI-4HD2D (CASIOPEIA Extension) |
| Project Number | E.02.14 |
| Project Manager | The Innaxis Foundation and Research Institute |
| Deliverable Name | Platform extension report V1 |
| Deliverable ID | D2.1 |
| Edition | 00.00.01 |
| Template Version | 03.00.00 |
| Task contributors | |
| The Innaxis Foundation and Research Institute.  University of Westminster | |
|  | |
| Abstract  This document is an intermediate 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 will be updated at the end of the project by the second – and final – version. | |

Authoring & Approval

|  |  |  |
| --- | --- | --- |
| Prepared By - Authors of the document. | | |
| Name & Company | Position & Title | Date |
| Irune Lansorena / The Innaxis Foundation and Research Institute | Consortium Member | 06/05/2015 |
| Alberto Blanch / The Innaxis Foundation and Research Institute | Consortium Member | 06/05/2015 |
| Luis Delgado / University of Westminster | Consortium Member | 06/05/2015 |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| Reviewed By - Reviewers internal to the project. | | |
| Name & Company | Position & Title | Date |
| Alberto Blanch / The Innaxis Foundation and Research Institute | Consortium Member | 06/05/2015 |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| 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 | 06/05/2015 |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| 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 | 06/05/2015 |  | INNAXIS | New Document |

Intellectual Property Rights (foreground)

This deliverable consists of SJU foreground.

Table of Contents

1 INTRODUCTION 5

1.1 Purpose of the document 5

1.2 Intended readership 5

1.3 Inputs from other projects 5

1.4 Glossary of terms 5

1.5 Structure of the Document 7

2 Agent-based model design overview 8

2.1 Introduction 8

2.2 Architecture design rationale 8

2.3 System architecture extension 9

2.3.1 Agent model extension 15

2.3.2 AMAN Processes 17

2.3.3 Cost index calculation processes 20

3 Agent-based model implementation overview 25

3.1 Agent model and simulation engine extension 25

3.1.1 Modelling new agents 25

3.1.2 Legacy agents 26

3.1.3 Communication among agents 28

3.1.4 Generating new indicators 30

3.2 Relational Data Model Extension 32

Appendix A CS3SESARImprovedFlight JavaBean 33

Appendix B CS3SESARImprovedAircraft JavaBean 36

Appendix C DCI4HD2D.application.xml 38

Appendix D Aman.agent.xml 44

Appendix E ArrivalManagement.capability.xml 45

Appendix F Aircraft.java 47

Appendix G AgentType.java 48

Appendix H CS3DCI4HD2DSimulation.java 50

Appendix I Simulation.java 60

Appendix J Manager.agent.xml 63

Appendix K ManagerProtocolPlan.java 66

Appendix L CreateAgentsPlan.java 69

Appendix M MessageType.java 71

Appendix N RequestArrivSlot.java 73

Appendix O IndicatorType.java 76

Appendix P CreateIndicatorsPlan.java 77

Appendix Q Pre-processing: flight plans inbound and outbound data manipulation and ingestion. 79

4 References 98

List of tables

Table 1 - ACL message parameters 29

Table 2 - DCI-4HD2D 31

Table 3 - Indicator table (physical data model) 32

List of figures

Figure 1 – FIPA compliant architecture 9

Figure 2 – JADEX-based MAS architecture. 10

Figure 3 - General entity-relationship diagram to be extended. 11

Figure 4 - Extended entity.relationship diagram (DCI-4H2D data-model design) 13

Figure 5 - Agent interaction diagram 15

Figure 6 - AMAN processes sequence diagram 18

Figure 7 - CS3 processes 21

Figure 8 - Inbound aircraft CI calculation process in DCI-4HD2D 22

Figure 9 - Outbound aircraft CI calculation before EOBT 23

Figure 10 - Outbound aircraft CI calculation after EOBT 24

Figure 11 – FIPA Agent communication specs 29

# INTRODUCTION

## Purpose of the document

This document shows the current ongoing work and status of CASSIOPEIA agent-based framework extension. This document is divided into two sotware engineering phases: agent model extension design and corresponding implementation. This document should help in understanding the overall process carried out to add, extend, or change different components. Thus, this document is expected to be useful to future policy-makers or operational users, or even to agent experts in understanding and quantifying workload in the analysis of new case studies.

**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 pointed out (Caire, G et.al, 2002), most of them fail in their attempts to adapt object-oriented methodologies to agent-based design and implementation. During CASSIOPEIA, Innaxis contributed 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; we decided to follow this line in DCI4HD2D project extension.

**Context**

This agent framework extension is placed under DCI4HD2D project, assessing DCI in 4HD2D context via CDM.

## Intended readership

This document is a technical agent design and implementation document under an extension project. However, for a better understanding of ongoing work and for simplicity many Agent Oriented Engineering artifacts have been avoided (AUML). Instead, the well-known software engineering Unified Modeling Language™ (UML) notation is used for more relevant components/artifacts in current extension. For further consultation about UML notation see: [www.uml.org](http://www.uml.org/).

## Inputs from other projects

This project is the extension of project E.02.14 CASSIOPEIA, and as such, many components will be related. However, for readability purposes, this document is self-contained.

## Glossary of terms

|  |  |
| --- | --- |
| Term | Definition |
| 4HD2D | Four Hour Door to Door |
| ABM | Agent-based Modelling/Model |
| ABS | Agent-based Simulation |
| ADF | Agent Definition File |
| AMAN | Arrival Manager |
| AOP | Agent-Oriented Programming |
| ATM | Air Traffic Management |
| AUML | Agent Unified Modeling Language |
| BDI | Beliefs Desires Intentions model |
| CDM | Collaborative Decision Making |
| DA-CDM | Deferred Acceptance Collaborative Decision Making |
| DCI | Dynamic Cost Index |
| DMAN | Departure Manager |
| EATM | European Air Traffic Management |
| EIBT | Estimated In-Block Time |
| EOBT | Estimated Off-Block Time |
| EOP | Environment-Oriented Programming |
| EPTI | Estimated Passing Time over IAF (Initial Approach Fix) |
| FIPA | The Foundation for Intelligent Physical Agents. It is a IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies. |
| FL | Flight level |
| FPL | Flight PLan |
| FRA | Free Route Airspace |
| IAF | Initial Approach Fix |
| IATA | International Air Transport Association |
| ICAO | International Civil Aviation Organisation |
| JADE | Agent middleware platform/framework |
| JADEX | A BDI-Agent System Combining Middleware and Reasoning |
| KPI | Key Performance Indicator |
| MAS | Multi-Agent System |
| MCT | Minimum Connecting Time |
| MTOW | Maximum Take-Off Weight |
| MTT | Minimum Turnaround Time |
| NM | Network Manager |
| NM | Nautical Miles |
| NOP | Network Operating Plan |
| OOP | Object-Oriented Programming |
| OQL | Object Query Language |
| Pax | Passenger(s) |
| PTI | Passing Time over IAF (Initial approach fix) |
| RBS | Ration by schedule |
| SESAR | Single European Sky ATM Research Programme |
| SJU | SESAR Joint Undertaking |
| SQL | Structured Query Language |
| TOC | Top Of Climb |
| TOT | Take-Off Time |
| UPM | Universidad Politécnica de Madrid |
| URL | Uniform Resource Locator |

## Structure of the Document

This document is structured as follows:

* Section 1 covers the design methodology process and software prototype extension context and scope.
* Section 2 presents agent model design overview, explaining different architecture related components' extension.
* Section 3 introduces agent model implementation overview taking into account Section 2 architectural decisions.

# Agent-based model design overview

## ****Introduction****

During CASSIOPEIA project, we follow a component-oriented approach for extending JADEX current architecture (Service Component Architecture or SCA). This component programming model, on top of JADE, allowed us to design our first three case studies (and correponding scenarios) and current extension (DCI4HD2D) as reusable components that helped us in dealing with case studies' complexity.

## ****Architecture design rationale****

DCI 4HD2D operational paradigm is currently being supported by the *legacy* architecture of CASSIOPEIA (see figure).

1. Agent Model
2. Simulation
3. DataBase
4. Visualization

As expected, this allowed us to reuse features from the previous architecture while providing new functionalities.Thanks to the intended abstraction design carried out during CASSIOPEIA project (e.g., three levels of generality: domain-independent, domain-specific and case-specific, see figure 1) we are mainly dealing with case-specific components, reusing as possible domain-independent and domain-specific artifacts as building blocks.

All of our architectural decisions aim to provide an agent-based framework that can model open/dynamic agent network for decision-making. Standardization and service discover/binding at runtime is a key to achieve that (see FIPA compliant architecture elements in Figure 1).

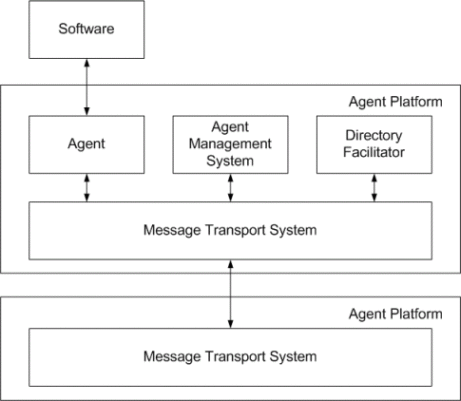


Figure 1 – FIPA compliant architecture

Utilizing a visualization tool in Java is out of the scope of the project extension. Thus, no modifications have been added. However, as a result of using the JADEX simulation toolsuite, we can control simulation features in ABS as well as monitorize agent communication protocols from JADE communication infrastructure (Message Transport System) in Multi Agent Systems.

Regarding the database, and due to the functional and non-functional requirements of DCI-4HD2D case study, we have been working on current database data preprocessing in order to model different expected SESAR-related improvements. Due to the nature of data - mainly **static** inputs for the simulation (working memory) - this preprocessing has not been carried out during simulation, but before it. The relational agent model has been extended to introduce new operational concepts, decision entities (agent instances), and KPIs to the model.

This process will be explained in the next sections.

## ****System architecture extension****

We are focusing on the following 3 component blocks during architecture extension (based on Figure 2):

1. Agent model extension.
2. Simulation engine.
3. Working memory and relational data model extension.

##### Agent model extension design

There have been several challenges to deal with during the agent-based design process. Some challenges include: potential aggregation of behavior (i.e, collective agents), which level of abstraction fits better our case study (micro operational, meso, tactical or macro level) or some parsimony-related challenges (e.g., number of agents, complexity in ATM agents' reasoning model from simple rules to more sophisticated agents).

Based on successful methodology and expertise gained on designing previous case studies, these basic steps have been followed:

1. Identify each decision maker in the ATMS case study (new or reusable).
2. Decide whether that decision maker is an agent or a simple feature from the environment (beliefs or a communication/coordination channel for agents, such as NOP).
3. If that decision maker is an active component or agent, define its "**role**" during the simulation and propose different algorithms/processes for modelling its behaviour. Note that under CASSIOPEIA and DCI4HD2D extension, roles are composed mainly by knowledge of the world (beliefs), motivations (goals), permissions, organizational relationship and interaction protocols identified by domain experts.
4. Establish FIPA-compliant interaction protocols and strategies.
5. Define agent mental model based on practical reasoning of BDI.

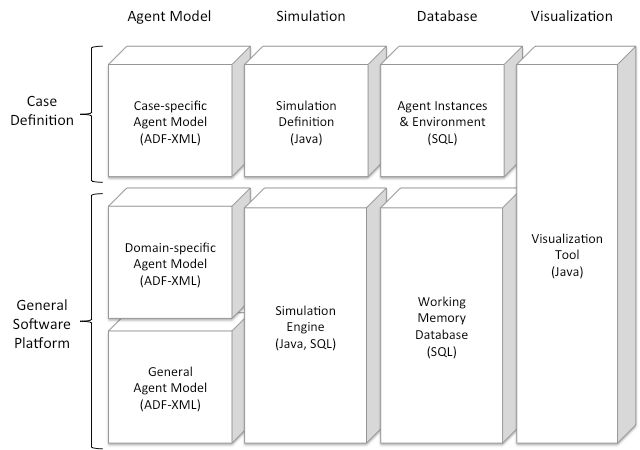


Figure 2 – JADEX-based MAS architecture.

##### Extending Relational data model

During CASSIOPEIA project, we designed and created a relational database (see Figure 3), both for domain-dependant modeling and case-specific features, attributes, and relationships. These entities and attributes (tables and columns in our physical relational model) represent agent model inputs, intermediate results (simulation historical data), and outputs (KPI-s for ATM performance evaluation).

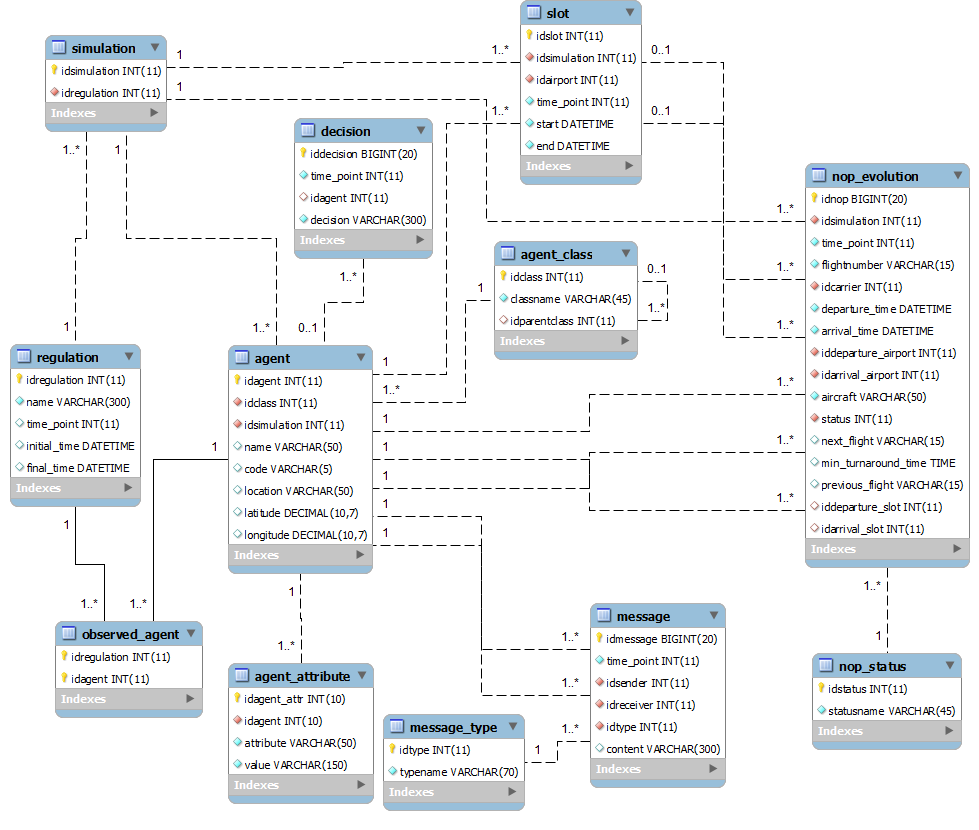


Figure 3 - General entity-relationship diagram to be extended.

Before extending current relational data model, and due to some anomalies detected on CS3 under CASSIOPEIA, a **data validation** process has been applied. We consider this process as an important starting point since the dynamic of the model (performance metrics as well) depends on a realistic scenarios simulation (operational and modeling consistency on data).

The following steps were taken to standardize the dataset:

1. Since the flight plans started at the first waypoint and ended at the last waypoint, there were two segments missing: from departure aerodrome to first waypoint, and from last waypoint to arrival aerodrome. In some cases, these missing segments led to some inconsistencies, such as some great circle distances between origin and destination airports being longer than the sum of the segments on the flight plan. In order to solve this, the great circle distances were calculated from the first waypoint to the last. This is more accurate since aircraft will always have to depart and arrive in the runway direction.
2. In order to avoid other problems, such as missing segments, the distance for each segment was recalculated considering the end point and the beginning of the following segment. After revision of the data, apparently only one segment was missing which could be constructed from information from previous and next segment.
3. The radius of the earth considered in the Harvesine formula applied is 3440.0 NM.

As established in D1.1 DCI4HD2D DataSet Management and case study design, the following SESAR Operational Improvements should be implemented:

* SESAR Trajectory Improvements (free routing).
* Ground Improvement Processes.
* Aircraft weight, flight level and fuel modeling**.**

Due to the nature of the current CASS architecture and requirements of the case study extension, aforementioned changes have been done as a data preprocessing. The main reason is the nature of data requirements. All data to be cleaned, generated, and extended to tackle new SESAR and Ground Improvements are input data.

Thus, relation model extension and data ingestion were carried out before simulation starts/runs (and not during the simulation), leading to the following new entity relationship diagram shown in figure 4.

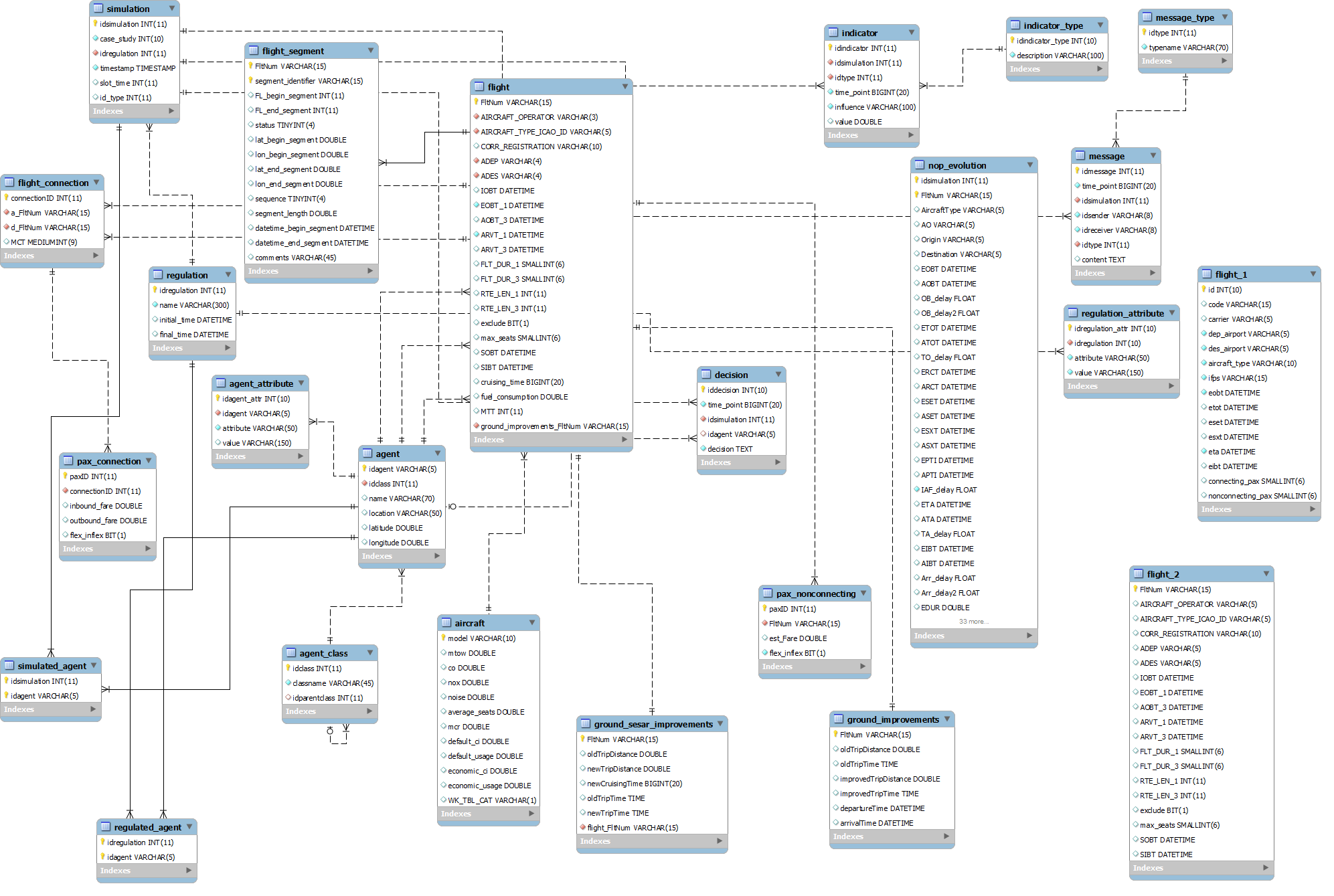


Figure 4 - Extended entity.relationship diagram (DCI-4H2D data-model design)

However, SESAR improvements modelling not only requires relational model extension, but also changes in all tiers of CASSIOPEIA MultiAgent System:

###### Adding new conceptual entities to simulation engine:

CS3SESARImprovedFlight: This Java entity represents new flight plan concept incorporating different SESAR and Ground improvements. This class includes attributes for describing improved flight plans such as newTripDistances and times. The code for this class is shown in Appendix A.

CS3SESARImprovedAircraft: This Java entity describes the fuel model to be applied: different coefficient, FL, aircraft type, distMin, distMax, Smin and resulted consumption values (kg/km). This entity is shown in Appendix B.

###### Extending current persistent tier:

* The aforementioned conceptual entity has its representation in our current ABM database. For that, new tables were created (ground\_improvements, ground\_sesar\_improvements).
* Adding a new attribute for original flight plan entity (CS3Flight.java (subclass of Flight.java part of domain-specific model) and update corresponding table (flight) : cruising\_time.

For more details on data ingestion and relational model implementation see data ingestion on *3 Agent-based model implementation overview.*

### Agent model extension

The following figure shows the interaction among the different agents in the model:

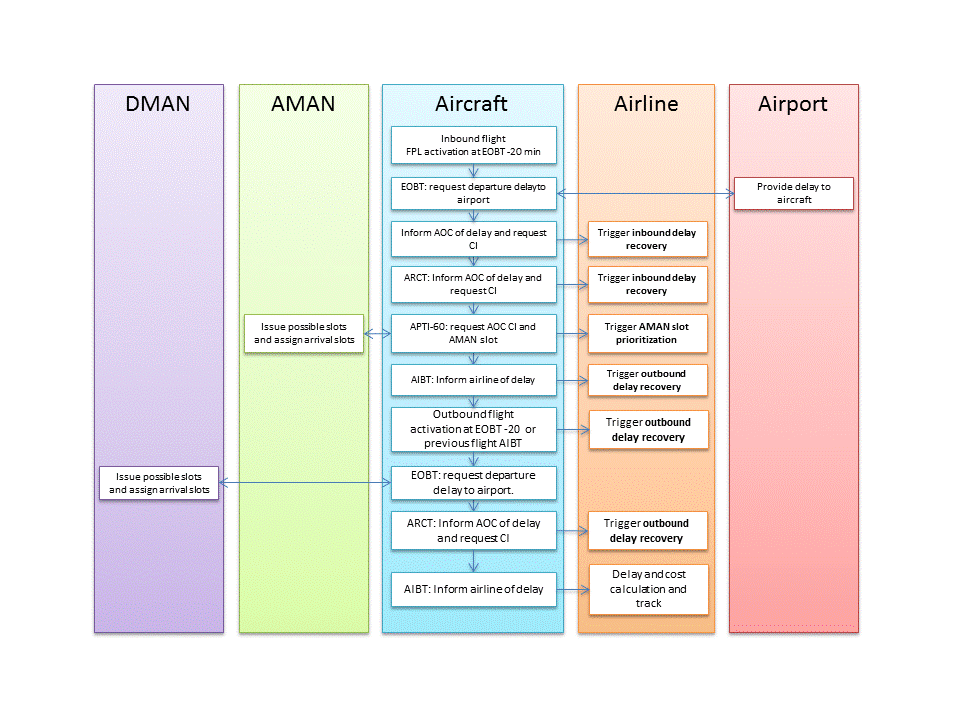


Figure 5 - Agent interaction diagram

**Description of processes**

**Inbound FPL activation**

Twenty minutes before the estimated off-block time of an inbound flight, this flight's flight plan will become activated. In real life this means that the flight is taking place and that the flight plan is sent through the IFPS to all relevant ANSPs for flow and capacity management purposes.

**Inbound departure delay**

At the inbound's estimated off-block time, the aircraft will contact the departure airport to request the departure delay. The relevant airport, simulated as a single agent, will provide the aircraft the departure delay based on a stochastic function that depends on the amount of annual movements at that airport.

**Initial CI calculation**

After receiving the departure delay, the aircraft will inform the airline of the given delay and request the cost index for the flight.

**ARCT CI calculation**

When the aircraft reaches cruise level, it will resend the actual delay to the airline operations centre to recalculate the cost index for the flight.

**AMAN negotiation**

One hour before reaching the initial approach fix of the hub airport studied, the aircraft will contact the approach manager to receive approach slot options. This process is further described in section *2.3.2 AMAN Processes*.

**Inbound arrival**

At the inbounds' arrivals, the next flight with the same aircraft will be activated. Also, aircraft pending to receive connecting passengers will have the actual time for those passengers to connect. The actual delay, emissions and cost information will be logged for performance metrics.

**Outbound FPL activation**

Twenty minutes before the estimated off-block time of an outbound flight, this flight's flight plan will become activated. Outbound delay recovery will be triggered which will make the AOC calculate the "Wait for pax" time and the cost index for the cruise phase of the flight.

**Inbound departure delay**

At EOBT, the flight will contact the departure manager to receive the departure delay. The DMAN will search for the next available slot in the departure queue and send that slot to the departing aircraft.

**ARCT CI calculation**

When the outbound flight reaches cruising level, it will contact its AOC to recalculate the cost index. This is necessary since there may be additional delay due to the departure queue plus additional vectoring delay simulated stochastically.

**Outbound arrival**

Upon arrival of the outbound flight, the actual delay, costs and emissions need to be saved for performance metrics.

### AMAN Processes

As defined in D1.1, the AMAN will implement a deferred acceptance CDM (DA-CDM) algorithm based on (Arruda Junior, Weigang, & Nogueira, 2014). The AMAN uses the preferences stated by the airlines regarding the available slots to assign them. In case of conflict with similar preferences, RBS prioritisation will be used.

The AMAN manages a list of slots available and assigned. It will also have a list of arriving flights to the airport with their estimated times of arrival.

The following figure depicts the sequence of processes sent among the different agents during the AMAN approach sequence request:

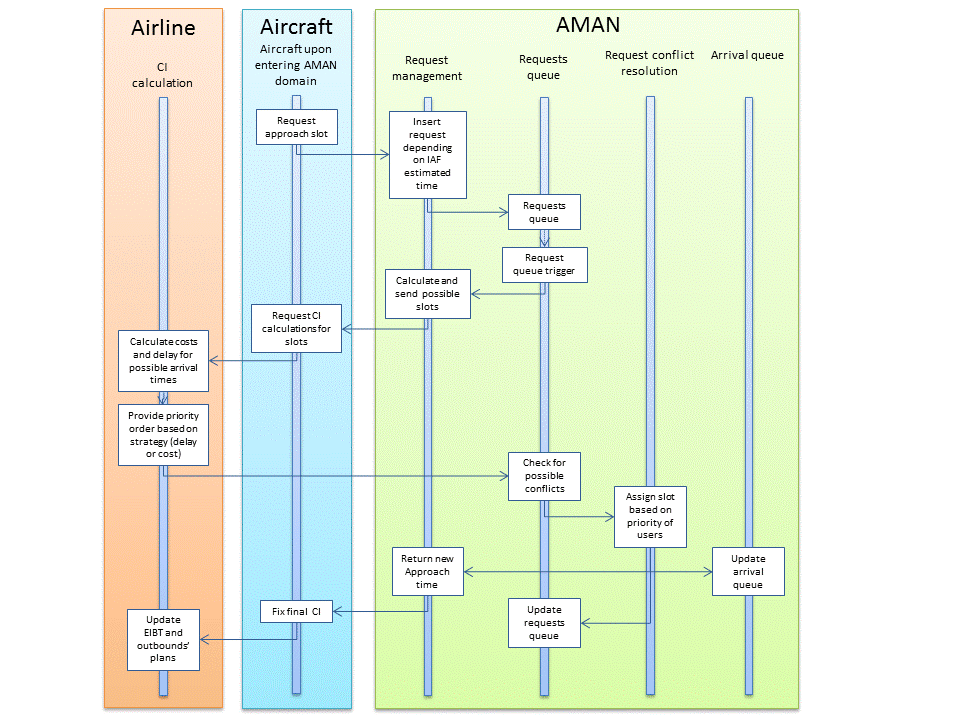


Figure 6 - AMAN processes sequence diagram

**Steps and messages**

1. Aircraft requests approach slot:   
   Based on current extended AMAN operations taking place in London Heathrow approach (350NM radius), once an aircraft is within one hour (based on current airspeed) of the IAF (initial approach fix) it will send the request to the AMAN for the approach slot. This message is the following:  
   <"slot request"; Flight ID; Tci; Tmin; Tmax>   
   where Flight ID is the flight number;  
   Tci is the IAF estimate based on current CI.   
   Tmin is the earliest time the flight can reach the IAF at top speed;  
   Tmax is the latest time the flight can reach the IAF without holding.
2. The AMAN will receive the message and insert the request into the Request queue based on the estimated approach time under current airspeed.
3. The AMAN must assign approach slots for those aircraft in the request queue. Whenever a flight enters the queue, this process is triggered. Or, if there is more than one flight on the list, the flight starts following the queue.
4. The AMAN will check the slots available for each flight within the range [[Tmin, Tmax](https://research.innaxis.org/pages/createpage.action?spaceKey=CAS&title=Tmin%2C+Tmax&linkCreation=true&fromPageId=34974744)], and sends the following message to the flight:  
   <"slot availability"; #slots; hh:mm; hh:mm; hh:mm; hh:mm; hh:mm; hh:mm; hh:mm; hh:mm; ...>  
   Where #slots is the number of slots available, and hh:mm is the timestamp of each available slot.  
   If all the slots in the range have already being assigned, the earliest available slot will be selected for the flight. If the aircraft cannot land in the 30 minutes after arriving to the airport then the earliest slot should be given to it and a minimum fuel consumption speed selected by the flight.
5. The flight will resend the message to the airline operations centre requesting priority sequence:  
   <"AMAN slot priority request"; Flight ID;  hh:mm; hh:mm; hh:mm; hh:mm; hh:mm; hh:mm; hh:mm; hh:mm; ...>
6. The airline will run the CI calculation for all slots available in the message received. Calculating pax delay and cost for each slot.
7. The airline will sort the slot priority list based on the strategy scenario (reduce pax delay or reduce cost) and send the following message to the AMAN:  
   <"AMAN slot priority order"; Flight ID; 2; 1; 3; 4; 5; 6; ...>  
   Where the numbers after the flight ID indicate the order of priority of the slots provided in the previous message.
8. The AMAN will receive the prioritisation of the slots available for the flight. However, it should wait to receive the requests of other flights that might be interested in the same slots so it can solve possible resource conflicts. For this reason, the AMAN will check the list of demand at the airport and check if there are flights in the window of slots submitted by the flight that have not sent their priorities yet.
9. If there are no more flights for those slots, it will solve the potential conflict with other flights, assign them their slots, and send that information to the flights.
10. If there are still some flights pending to send their requests, it will add the flight to the slot assignment queue with a timer of X minutes.

The AMAN will reply to the aircraft with the assigned slot:  
<"AMAN Slot reserved"; Flight ID; hh:mm>  
The AMAN will remove the flight from slot assignment queue.

1. The aircraft will change the CI based on the slot received by the AMAN and inform the airline of the new trajectory changes:  
   <"New AMAN slot"; Flight ID; hh:mm>
2. The airline will update its plan, updating new EIBT and related outbounds' CI.
3. If a flight has not been assigned a slot after the x minutes of waiting then the AMAN will solve the conflicts of slots for that flight and send the slot assigned to the aircraft. By doing so a flight will not have to wait more than X minutes to get its slot assignment. On the other hand, time is given to flights coming from distances closer than the radius of the AMAN to submit their requests.

### Cost index calculation processes

#### Inbound aircraft

##### Case Study 3 code

In order to implement the new requirements to the code, it was necessary to analyse the existing code to be able to discern how these requirements should be implemented.

The figure below shows the relationship between the different methods of the model to calculate the cost index and the related indicators.

There are two methods in the far right side which seem to be doubled, since the method to calculate hard and soft costs has already been defined.

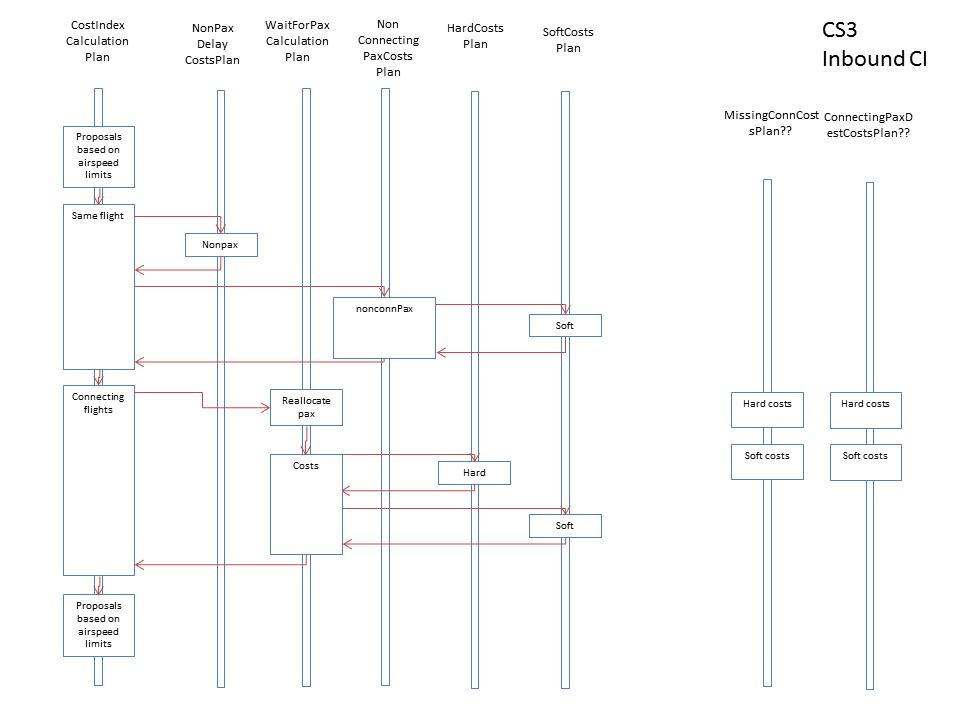


Figure 7 - CS3 processes

##### DCI-4HD2D

For the current project requirements, the following methods have been defined:

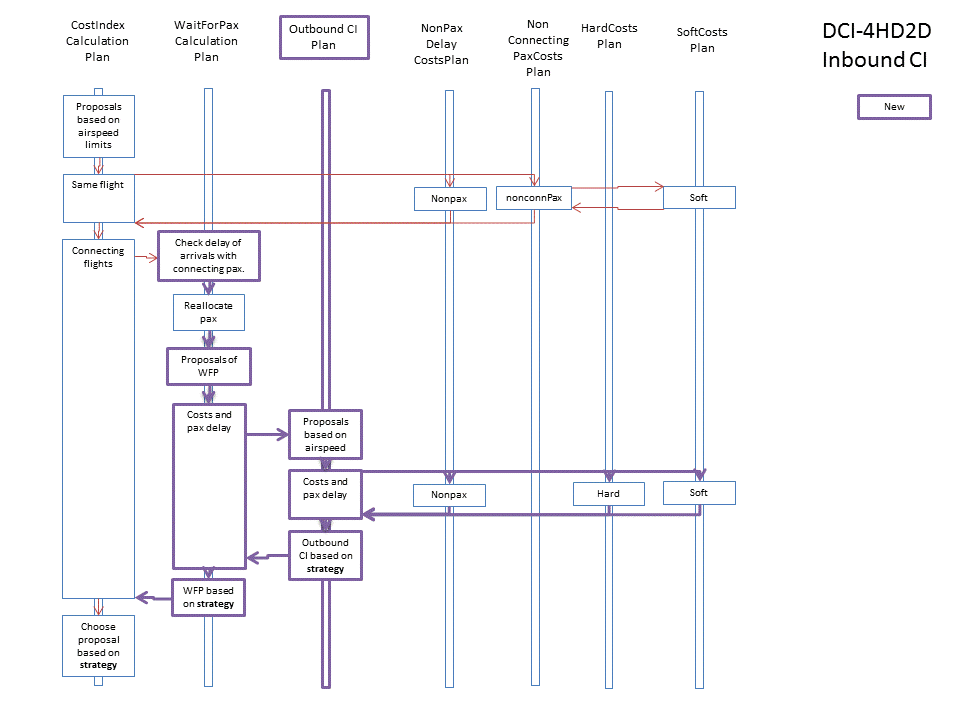


Figure 8 - Inbound aircraft CI calculation process in DCI-4HD2D

As can be observed in the figure 8, the cost index of the outbound flights is also calculated when the inbound cost index is calculated. For each inbound flight, 7 or 11 airspeed settings are calculated, depending on the scenario. For each airspeed setting, outbound flights are allowed to wait for the inbound or depart as scheduled (or wait for some of the late inbound flights). And for each departure time, also 7 or 11 airspeed settings are calculated. Each inbound flight has an average of 12 outbound flights with connecting passengers. Assuming that half of the flights are affected in a delay, there would be an average of 726 CI calculations for a single inbound flight calculation.

Additionally, the new process is able to choose "Wait for pax" time and CI based on the strategy selected in the scenario.

#### Outbound aircraft

For outbound aircraft, there are two different CI calculation processes. If the aircraft has not departed, it needs to calculate thewait for pax (WFP) time, and the CI for each WFP. The figure below shows the sequence of this process:

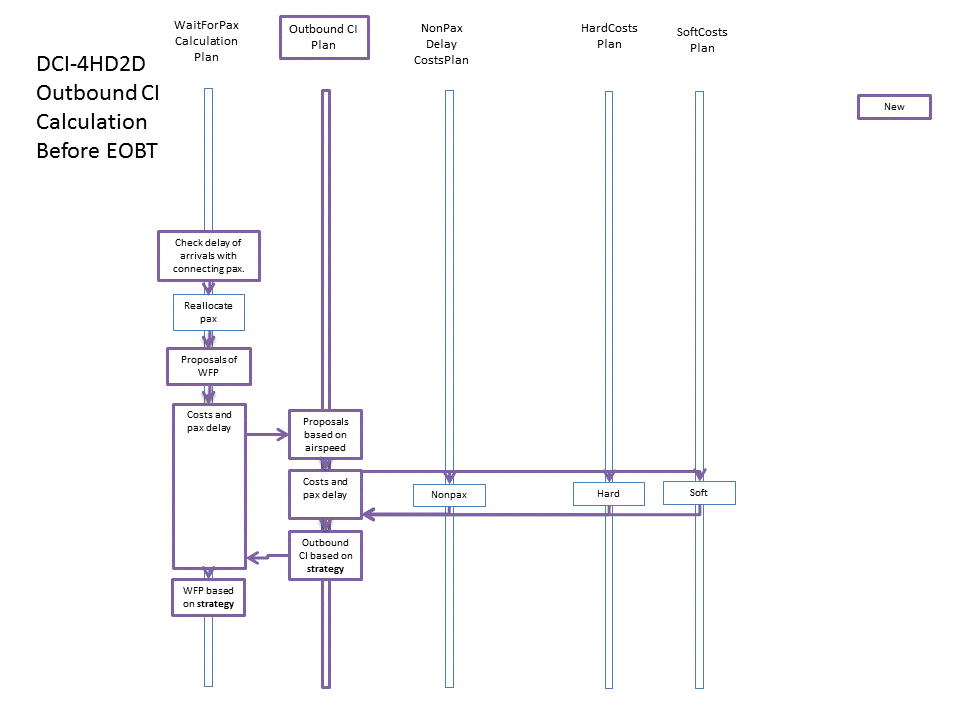


Figure 9 - Outbound aircraft CI calculation before EOBT

When the outbound aircraft has departed, the CI calculation process is executed as shown in the figure below:

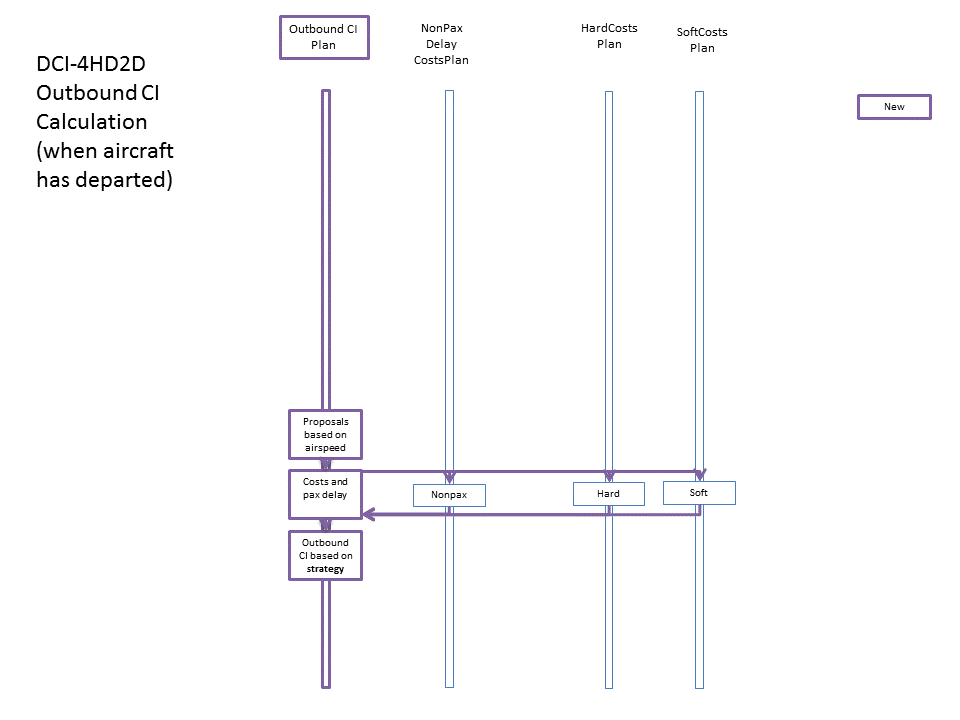


Figure 10 - Outbound aircraft CI calculation after EOBT

# Agent-based model implementation overview

In order to be consistent with legacy architecture, we are working on top of same building blocks as design process during the implementation DCI4HD2D extension process (agent model, simulation engine, relational data model).

## Agent model and simulation engine extension

In order to model the DCI4HD2D paradigm, additional agents and environmental components are needed. Furthermore, new strategies for airlines operating with DCI are needed, as well as new messages and interactions protocols. New features of the model and corresponding implementation are detailed below.

### Modelling new agents

The following new decision makers (agents) are part of our model:

* AMAN
* DMAN

Since JADEX framework follows a SCA (Service Component Architecture), each of above new agent will be modeled as an **active** component, a first class entity able to act **autonomously** and have its own thread of control.

Adding new agents and/or environmental components to the model follows declarative approach (XML files as an agent-based metamodel). This abstraction is a great feature, since we are able to provide a flexible parametrization of scenarios (configurations), enabling the analysis of multiple behavior and/or strategies for each agent or the system as a whole.

Before modeling each agent, we specified our meta-model (scenarios, configurations and arguments). These are mainly simulation control features. This metamodel can be seen in Appendix C.

As shown in the metamodel, some GUI components have been introduced as inputs for the model. Most of them remain from CS3. However, we introduced <simulationType> in order to enable policy maker to decide which scenario is analyzing:

* Baseline: This scenario works with original FlightPlan data from the time window (00:00:00 UTC-23:59 UTC from ZRH inbound and outbounds).
* GroundImprovements: In this case, freerouting has been modeled in order to show the deviation from baseline scenario through previously define KPI-s.
* SESAR+Ground improvements: This scenario works with a pre-processed data in order to model SESAR improvements under ground improvements.

This argument, in contrast to configurations, has no default value; it is configurable by operational experts fulfilling a parameter from JCC (Jadex Control Center).

Moreover, different configurations are controlled (S0...S11) selecting options via a combobox from JCC GUI. Each new configuration is composed by a set of arguments.

Once established the case study as a whole, we aimed to introduce new agents (AMAN/DMAN) and new behaviors for existing ones (Aicraft, AOC, NM).

The process of introducing a new agent to CASSIOPEIA ABM, involves a hybrid language process:

1. Agent Definition/Parametrization through Agent Descriptor File or ADF   
   This file is mostly static. It defines the agent type (using JADEX semantic for beliefs, desires, and goals) to be instantiated during the case study simulation and initial state (initialized at runtime).   
     
   Agent definitions' are XML, rule-based notation: jadex expression language, jadex condition language.  
     
   Note that for future framework users, it is important to understand Agent Model and Agent Runtime concepts as well as differentiate domain specific agent model and BDI Agent Metamodel. As an example, the AMAN agent XML definition is shown in Appendix D.

2. Adding "behavioral intelligence" to the agent.  
  
For that, we are currently implementing different motivational states for AMAN (DMAN as well), introducing new goals (e.g., serve slots and optimize arrivals following a prioritisation of the departure flights and the departure slots - see Appendix E), or trying different configurations using different goals for a sensitivity analysis.   
Another key feature is modeling AMAN's/DMAN's intentions implementing different plans (Pure Java Classes) or sequence of actions to achieve his objectives or react to Airlines needs/request (arrivals management). These plans have been designed providing different algorithmic solutions (see Agent extension in 3 Agent-model Design Overview).

We decided to model AMAN and DMAN as **agents**and not simply as an object part of the environment mainly because we identified many of the features of a classical agent in MAS community for our AMAN/DMAN decision maker (M. Wooldridge, 1999): autonomy (their own thread of control), proactivity, reactivity, and sociability. Both AMAN and DMAN agents are rational decision maker (bounded) following BDI model for human decision making (M.Bratman, 1987).

### ****Legacy agents****

There are synergies between agents in CS3 and DCI4HD2D case study. This is why we kept most of agents from CS3. These are legacy systems from CS3. However, we modeled different beliefs taking into account in which scenario we were working on. The agents considered are the following:

* Aircraft agent.
* Airline agent.
* Airport agent.

Most legacy agents' definitions remain as they were during CASSIOPEIA's CS3. Each agent has its own conceptual representation (Java class/bean as Aircraft.java below) with its own inherited attributes from General ABM Agent class (encapsulation of agents attributes, flights (beliefs) from the environment and, most importantly, a reference for ADF (agent definition path). An example is shown in Appendix F.

However, several BDI model related changes are being implemented. Firstly, CostIndexingCapability has changed in order to give legacy agents the capability to **perceive** new elements from the environment modeled in current CASSIOPEIA extension (Appendix H). The code describing the agent types is shown in Appendix G.

Furthermore, several changes were implemented in some legacy agents' capabilities (e.g., CostIndexingcapability.xml) to enable the agent to achieve and/or perform new goals. Achievement of these new goals usually requires communication and coordination among agents.

**Modelling ATM environment via CS3DCI4HD2DSimulation**

In spite of the fact that JADEX currently supports ABS-related environmental features (see above), we decided to follow our approach during CASSIOPEIA and model a more realistic environment. Apart from that, the rationale for not modeling environment in a NetLogo,SeSAm or Repast style is the fact of there are still several limitations in JADEX environmental support (following listed).  On the one hand, we do not follow an environment-oriented agent programming (environment as a first class for agent communication), we follow message-based programming instead. On the other hand, visualization is out of the scope of the project.

Finally, the most important reason for not modelling based on EnvSupport is the fact that we will not be able to deploy our agent-based model in a real environment in a distributed fashion, limiting our scalability options.

Thus, in our case, drawbacks related to modeliing environment through EnvSupport are stronger than benefits offered by JADEX (at least up to now).

**Features:**

* Declarative specification of the environment as application space
* Model definition consisting mainly of space objects, tasks and environment processes in a 2d or 3d grid or continuous world
* Agent-environment interaction via customizable percepts and actions
* 2d and 3d visualization of the environment, its objects and agents, including possibilities for animation, etc.
* Customizable space execution with built-in support for continuous and round-based execution semantics
* Highly extensible with a lot of ready-to-use components for frequent use cases

**Current Limitations:**

* No direct manipulation of the environment from the user interface
* No integrated collision detection
* Applications with EnvSupport cannot be distributed over several platform nodes

Thus, we model initial network state and case study/simulation dependant parameters through CS3DCI4HD2DSimulation class (Appendix H). This class extends the general ABM from CASSIOPEIA project (see Appendix I).

##### Enabling new agents ACTIVATION

In order to ensure agent-based model consistency, all agents are instantiated and activated at runtime by the manager agent. This emulates a "factory pattern" in which manager agent manages the creation of all agents of the simulation through its "cmscap" capability and cmscap.cms\_create\_component achieve goal by default and create\_agents perform goal. As an example, the manager agent is shown in Appendix J.

Note that in manage.agent.xml ADF, we enable manager (it is the only agent) to see initial world state (e.g., ATM network state and simulation parameters) defining adding to its initial beliefs created CS3CI4HD2DSimulation class (simulationDCI4HD2D belief).

Apart from that, manager agent will be responsible on creating new indicators of DCI4HD2D case study as well as creating/activating all simulated agents (see ManagerProtocolPlan.java in Appendix K). The creation of the agents is done through the class shown in Appendix L.

### Communication among agents

Following CASSIOPEIA platform agent-based design rationale, agent communication and/or interaction protocols are mainly based on standardized FIPA-compliant **A**gent **C**ommunicaton **L**anguage (ACL) messages [FIPA, 2015].  This approach provides to the platform openness as well as interoperability with future agent-based applications.

As the figure below depicts, messages define type of speech act theory-based, communicative acts and content language representations.

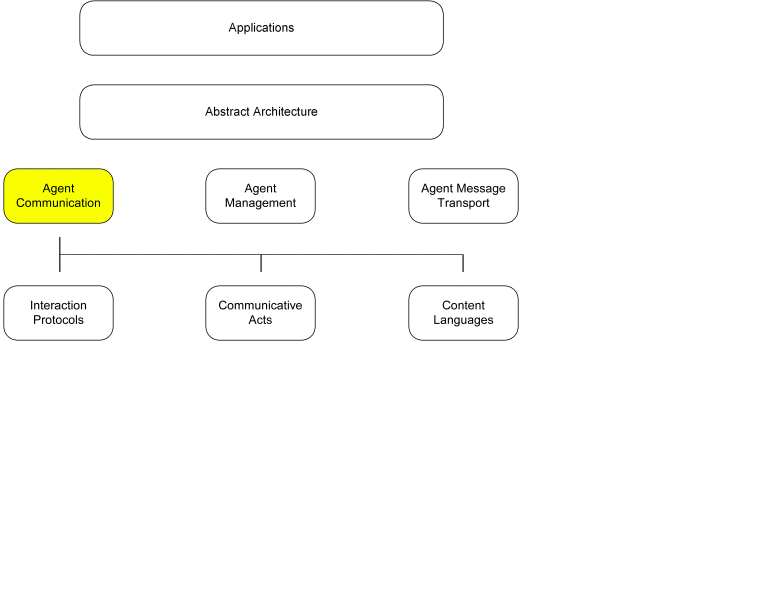


Figure 11 – FIPA Agent communication specs

JADEX represents messages as external events. Each agent is able to "handle" each type of events (defined following a declarative language XML in ADF). Thus, each agent will be able to send, receive or both at **runtime**.

For simplicity, during CASSIOPEIA, messages were encoded as native datatype String (<content> parameter class =String). However, the extension includes more complex message structures including domain dependent objects when necessary (as it is the case of a data structure containing different proposals - ordered by desirability - from airlines to AMAN agent). However, best practice is to avoid network overload and complex marshall and unmarshalling requirements. Messages are shown in Appendix M. Messages’ parameters are shown in the table below.

Table 1 - ACL message parameters

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Meaning** |
| performative | String | Speech act of the message that expresses the senders intention towards the message content. You can use the constants from jadex.base.fipa.SFipa.{ACCEPT\_PROPOSAL, AGREE, ...} |
| sender | IComponentIdentifier | The senders agent identifier, which contains besides other things its globally unique name. |
| reply\_to | IComponentIdentifier | The agent identifier of the agent to which should be replied. |
| receivers [set] | IComponentIdentifier | Arbitrary many (at least one) agent identifier of the intended receiver agents. |
| content | Object | The content (string or object) of the message. If the content is an object it has to be specified how the content can be marshalled for transmission. For this puropose codecs are used. Jadex has built in support for marshalling arbitrary Java beans via setting the language of the message to *jadex.base.fipa.SFipa.JADEX\_XML*. |
| language | String | The language in which the content of the message should be encoded. |
| encoding | String | The encoding of the message. |
| ontology | String | The ontology that can be used for understanding the message content. Can also be used for deciding how to marshal the content. |
| protocol | String | The interaction protocol of the the message if it belongs to a conversation. There are constants available for the predefined FIPA interaction protocols in *jadex.base.fipa.SFipa.PROTOCOL\_{REQUEST, QUERY, ...}* |
| reply\_with | String | Reply\_with is used for assigning a reply to a original message. The receiver of the message should respond to this message by putting the reply\_with value in the in\_reply\_to field of the answer. Unique ids can e.g. be generated via the method SFipa.createUniqueId(). |
| in\_reply\_to | String | Used in reply messages and should contain the reply\_with content of the answered message. |
| conversation\_id | String | The conversation\_id is used in interactions for identifying messages that belong to a specific conversation. All messages of one interaction should share the same conversation\_id. Unique ids can e.g. be generated via the method SFipa.createUniqueId(). |
| reply\_by | Date | The reply\_by field can contain the latest time for a response message. |

For tracking, future analysis, and interesting visualizations of simulation dynamics, each FIPA message has its own POJO to be stored in **message** and **message\_type** table (see extended Entity/Relationship). This is shown in Appendix N.

### Generating new indicators

For the assesment of new case studies and different policy instruments (scenarios) we designed different local and system-wide indicators (KPI). In order to generate those outcomes, we extended both our relational data model as well as agent-based model (mainly simulation engine).

The following table shows the different indicators used: passenger oriented, aircraft performance, eco-sustainability related indicators, and different metrics evaluating solution complexity.

Table 2 - DCI-4HD2D

| **Id** | **Indicator** | **Unit** | **Category** |
| --- | --- | --- | --- |
| 1 | Gate-to-gate passenger trip time | Minutes | Passengers performances |
| 2 | Door-to-door passenger trip estimation | Minutes |  |
| 3 | Societal cost estimation | Euros |  |
| 4 | Pax delay | minutes |  |
| 5 | Missed connections | Passenger |  |
| 6 | Flight delay | Minutes | Aircraft performances |
| 7 | Airlines cost | Euros |  |
| 8 | Hub airline cost | Euros |  |
| 9 | Hub airline cost variation with respect to optimum | Euros |  |
| 10 | Non-hub airline cost | Euros |  |
| 11 | Non-hub airline cost variation with respect to optimum | Euros |  |
| 12 | Speed variations incurred | Percentage | Complexity of the solution |
| 13 | Messages interchanged | Messages |  |
| 14 | Emissions | Tons CO2 | Eco-sustainability |

In order to model these new KPIs, several components have been changed. Since there is no new domain concept or entity, there was no need to extend current relational data model. However, an agent simulation engine is currently being extended for computation of these new indicators. Concretely, we introduced a new indicator type (see IndicatorType.java enumerator class in Appendix O).

Since the *Manager* agent has **system-wide**beliefs (inherited from CASSIOPEIA project), we decided to keep this central approach when generating new indicators in DCI4HD2D. In this way, Manager.agent.xml (Appendix J) keeps its goal for creating indicators (<performgoal name="create\_indicators" />). Most changes are currently being implemented in its corresponding behavioral class (CreateIndicatorsPlan.java, Appendix P). This plan accesses the current network state (through "flights" **set of beliefs**) and is responsible for creating the final state indicators.

All indicators generated at runtime (during simulation) will be stored in cassiopeia\_cs3 database for later analysis. Indicators are modelled as Persistence elements which implements save Statement method for adding a new indicator, simulationID, type, type\_point or timestamp in which the indicator is generated, influence and value.

Table 3 - Indicator table (physical data model)

|  |  |  |
| --- | --- | --- |
| **IDINDICATOR** | Identifier for the indicator | UNSIGNED INTEGER |
| IDSIMULATION | Identifier of the simulation that produces this indicator (limited to identifiers of SIMULATION table) | UNSIGNED INTEGER |
| IDTYPE | Identifier of the type of indicators created during the simulation (ordinal position of IndicatorType enum) | UNSIGNED INTEGER |
| TIME\_POINT | TimeStamp for indicator creation | BIGINT (20) |
| INFLUENCE | - | VARCHAR(100) |
| VALUE | The value taken by the indicator | DOUBLE |

For statistical summaries we are currently using Apache Commons Math (Apache, 2015) as shown in Appendix P.

## Relational Data Model Extension

During CASSIOPEIA project, agent-based model was supported by a relational database. This relational database not only enabled the abstraction of general agent-based models able to tackle with different ATM-related Case Studies, but also extended this model with new datasets in order to deal with new case studies, scenarios, or policy instruments. This is shown in Appendix Q.

1. CS3SESARImprovedFlight JavaBean

|  |
| --- |
| package simulator.environment;  import java.sql.Time;  import java.util.ArrayList;  import java.util.List;    /\*\*   \* @author IruneLansorena   \*   \* New entity incorporating different SESAR and Ground improvements. Fuel model added.   \*   \*/  public class CS3SESARImprovedFlight {        private List<Segment> segments = new ArrayList <Segment> ();      private double airportsDistance;      private int maxFL;      private int maxFLTotal;      private int inicioCrucero;      private int finalCrucero;      private String FltNum;      private String ADEP;      private String ADES;      private double oldTripDistance;      private Time oldTripTime;      private double newTripDistance;      private Time newTripTime;      private String AIRCRAFT\_TYPE\_ICAO\_ID;      private double consumption;      public double getOldTripDistance() {          return oldTripDistance;      }      public void setOldTripDistance(double oldTripDistance) {          this.oldTripDistance = oldTripDistance;      }      public Time getOldTripTime() {          return oldTripTime;      }      public void setOldTripTime(Time oldTripTime) {          this.oldTripTime = oldTripTime;      }      public double getNewTripDistance() {          return newTripDistance;      }      public void setNewTripDistance(double newTripDistance) {          this.newTripDistance = newTripDistance;      }      public Time getNewTripTime() {          return newTripTime;      }      public void setNewTripTime(Time newTripTime) {          this.newTripTime = newTripTime;      }      public int getInicioCrucero() {          return inicioCrucero;      }      public void setInicioCrucero(int inicioCrucero) {          this.inicioCrucero = inicioCrucero;      }      public int getFinalCrucero() {          return finalCrucero;      }      public void setFinalCrucero(int finalCrucero) {          this.finalCrucero = finalCrucero;      }      public int getMaxFLTotal() {          return maxFLTotal;      }      public void setMaxFLTotal(int maxFLTotal) {          this.maxFLTotal = maxFLTotal;      }      public int getMaxFL() {          return maxFL;      }      public void setMaxFL(int maxFL) {          this.maxFL = maxFL;      }      public String getFltNum() {          return FltNum;      }      public void setFltNum(String fltNum) {          FltNum = fltNum;      }        public double getAirportsDistance() {          return airportsDistance;      }      public void setAirportsDistance(double airportsDistance) {          this.airportsDistance = airportsDistance;      }      public List<Segment> getSegments() {          return segments;      }      public void setSegments(List<Segment> segments) {          this.segments = segments;      }      public void addSegment (Segment seg){          this.segments.add(seg);      }      public String getADEP() {          return ADEP;      }      public void setADEP(String aDEP) {          ADEP = aDEP;      }      public String getADES() {          return ADES;      }      public void setADES(String aDES) {          ADES = aDES;      }      public String getAIRCRAFT\_TYPE\_ICAO\_ID() {          return AIRCRAFT\_TYPE\_ICAO\_ID;      }      public void setAIRCRAFT\_TYPE\_ICAO\_ID(String aIRCRAFT\_TYPE\_ICAO\_ID) {          AIRCRAFT\_TYPE\_ICAO\_ID = aIRCRAFT\_TYPE\_ICAO\_ID;      }      public double getConsumption() {          return consumption;      }      public void setConsumption(double consumption) {          this.consumption = consumption;      }  } |

1. CS3SESARImprovedAircraft JavaBean

package simulator.environment;

/\*\*

 \* @author IruneLansorena

 \*

 \* New entity incorporating different SESAR and Ground improvements. Fuel model added.

 \*

 \*/

public class CS3SESARImprovedAircraft {

    //UoW coefficient fuel model based on FL, speed and aircraft model

    private double p1;

    private double p2;

    private double p3;

    private double p4;

    private double p5;

    private double consumption;

    private String type;

    private int FL;

    private int distMin;

    private int distMax;

    private double Smin;

    public CS3SESARImprovedAircraft() {

        super();

        // TODO Auto-generated constructor stub

    }

    public CS3SESARImprovedAircraft(double p1, double p2, double p3, double p4,

            double p5, double consumption, String type, int fL, int distMin,

            int distMax, double smin) {

        super();

        this.p1 = p1;

        this.p2 = p2;

        this.p3 = p3;

        this.p4 = p4;

        this.p5 = p5;

        this.consumption = consumption;

        this.type = type;

        FL = fL;

        this.distMin = distMin;

        this.distMax = distMax;

        Smin = smin;

    }

    public double getSmin() {

        return Smin;

    }

    public void setSmin(double smin) {

        Smin = smin;

    }

    public int getDistMin() {

        return distMin;

    }

    public void setDistMin(int distMin) {

        this.distMin = distMin;

    }

    public int getDistMax() {

        return distMax;

    }

    public void setDistMax(int distMax) {

        this.distMax = distMax;

    }

    public String getType() {

        return type;

    }

    public void setType(String type) {

        this.type = type;

    }

    public double getP1() {

        return p1;

    }

    public void setP1(double p1) {

        this.p1 = p1;

    }

    public double getP2() {

        return p2;

    }

    public void setP2(double p2) {

        this.p2 = p2;

    }

    public double getP3() {

        return p3;

    }

    public void setP3(double p3) {

        this.p3 = p3;

    }

    public double getP4() {

        return p4;

    }

    public void setP4(double p4) {

        this.p4 = p4;

    }

    public double getP5() {

        return p5;

    }

    public void setP5(double p5) {

        this.p5 = p5;

    }

    public double getConsumption() {

        return consumption;

    }

    public void setConsumption(double consumption) {

        this.consumption = consumption;

    }

    public int getFL() {

        return FL;

    }

    public void setFL(int fL) {

        FL = fL;

    }}

1. DCI4HD2D.application.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>    <!--        <H3>Case 4:  DCI4HD2D Simulator</H3>    -->    <applicationtype xmlns="        xmlns:xsi="        xsi:schemaLocation="<http://jadex.sourceforge.net/jadex>            name="DCI4HD2D" package="simulator">          <componenttypes>            <componenttype name="Airport"  filename="simulator/airport/Airport.agent.xml" />            <componenttype name="Airline"  filename="simulator/airline/Airline.agent.xml" />            <componenttype name="Aircraft"  filename="simulator/aircraft/Aircraft.agent.xml" />            <componenttype name="Manager"  filename="simulator/manager/Manager.agent.xml" />            <componenttype name="AMAN"  filename="simulator/aman/Aman.agent.xml" />            <componenttype name="DMAN"  filename="simulator/dman/Dman.agent.xml" />            <componenttype name="ImprovedFlight"  filename="simulator/SESAR/ImprovedFlight.agent.xml" />        </componenttypes>        <arguments>          <argument name="fuelprice"    class="float" >0.6f</argument>          <argument name="co2ratio"     class="float" >3.15f</argument>          <argument name="segmentid"    class="String">"FERDI\_BUPAL"</argument>          <argument name="airportid"    class="String">"LSZH"</argument>          <argument name="uncertainty"  class="boolean">true</argument>          <argument name="regulationid" class="int"  />          <argument name="simulationType" class="int"  />      </arguments>      <configurations>          <configuration name="S000">              <arguments>                  <argument name="regulationid">1</argument>              </arguments>                <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>            </configuration>            <configuration name="S001">              <arguments>                  <argument name="regulationid">2</argument>              </arguments>                <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>          </configuration>            <configuration name="S002">              <arguments>                  <argument name="regulationid">3</argument>              </arguments>              <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>            </configuration>            <configuration name="S003">              <arguments>                  <argument name="regulationid">4</argument>              </arguments>              <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>            </configuration>            <configuration name="S010">              <arguments>                  <argument name="fuelprice">0.9f</argument>                  <argument name="regulationid">1</argument>              </arguments>              <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>            </configuration>            <configuration name="S011">              <arguments>                  <argument name="fuelprice">0.9f</argument>                  <argument name="regulationid">2</argument>              </arguments>                <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>            </configuration>            <configuration name="S012">              <arguments>                  <argument name="fuelprice">0.9f</argument>                  <argument name="regulationid">3</argument>                </arguments>                <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>            </configuration>            <configuration name="S013">              <arguments>                  <argument name="fuelprice">0.9f</argument>                  <argument name="regulationid">4</argument>              </arguments>                <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>            </configuration>            <configuration name="S101">              <arguments>                  <argument name="regulationid">12</argument>              </arguments>              <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>            </configuration>            <configuration name="S102">              <arguments>                  <argument name="regulationid">13</argument>              </arguments>              <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>            </configuration>            <configuration name="S103">              <arguments>                  <argument name="regulationid">14</argument>              </arguments>                <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>            </configuration>            <configuration name="S111">              <arguments>                  <argument name="fuelprice">0.9f</argument>                  <argument name="regulationid">12</argument>              </arguments>                <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>            </configuration>            <configuration name="S112">                <arguments>                  <argument name="fuelprice">0.9f</argument>                  <argument name="regulationid">13</argument>              </arguments>                <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>            </configuration>            <configuration name="S113">              <arguments>                  <argument name="fuelprice">0.9f</argument>                  <argument name="regulationid">14</argument>              </arguments>              <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>            </configuration>                  <configuration name="S201">              <arguments>                  <argument name="regulationid">22</argument>              </arguments>                <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>            </configuration>            <configuration name="S202">                <arguments>                  <argument name="regulationid">23</argument>              </arguments>                <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>            </configuration>            <configuration name="S203">              <arguments>                  <argument name="regulationid">24</argument>              </arguments>                <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>            </configuration>            <configuration name="S211">              <arguments>                  <argument name="fuelprice">0.9f</argument>                  <argument name="regulationid">22</argument>              </arguments>                <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>            </configuration>            <configuration name="S212">              <arguments>                  <argument name="fuelprice">0.9f</argument>                  <argument name="regulationid">23</argument>              </arguments>                <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>            </configuration>            <configuration name="S213">              <arguments>                  <argument name="fuelprice">0.9f</argument>                  <argument name="regulationid">24</argument>              </arguments>                <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>            </configuration>          <!-- New configurations for DCI4HD2D paradigm strategies -->          <configuration name="DCI4HD2DS0">              <arguments>                  <argument name="airspeed">0.05f</argument>                  <argument name="strategy">cost\_efficient</argument>                  <argument name="fuelprice"    class="float" />              </arguments>              <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>          </configuration>          <configuration name="DCI4HD2DS1">              <arguments>                  <argument name="airspeed">0.03f</argument>                  <argument name="strategy">reduce\_pax\_delay</argument>                  <argument name="fuelprice"    class="float" />              </arguments>              <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>          </configuration>          <configuration name="DCI4HD2DS2">              <arguments>                  <argument name="airspeed">0.05f</argument>                  <argument name="strategy">reduce\_pax\_delay</argument>                  <argument name="fuelprice"    class="float" />              </arguments>              <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>          </configuration>          <configuration name="DCI4HD2DS3">              <arguments>                  <argument name="airspeed">0.03f</argument>                  <argument name="strategy">cost\_efficient</argument>                  <argument name="fuelprice"    class="float" />              </arguments>              <components>                  <component type="Manager"  name="manager" configuration="standard" master="true" />              </components>          </configuration>        </configurations>    </applicationtype> |

1. Aman.agent.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!--      <H3>Aman.</H3>      Common beliefs  -->  <agent xmlns="      xmlns:xsi="      xsi:schemaLocation="<http://jadex.sourceforge.net/jadex>      name="AMAN"      package="simulator.aman">      <imports>          <import>simulator.environment.Simulation</import>          <import>simulator.environment.Flight</import>      </imports>      <capabilities>          <capability name="ArrivalManagement"  file="simulator.aman.ArrivalManagement" />      </capabilities>      <beliefs>          <belief name="simulationDCI4HD2D" class="Simulation" argument="true" >              <assignto ref="ArrivalManagement.simulationDCI4HD2D" />          </belief>            <belief name="icao"     class="String" >              <assignto ref="ArrivalManagement.icao" />          </belief>            <beliefset name="flights" class="Flight" >              <assignto ref="ArrivalManagement.flights" />          </beliefset>      </beliefs>      <properties>          <!-- Output log -->          <property name="logging.level">java.util.logging.Level.ALL</property>          <property name="logging.level.exceptions">java.util.logging.Level.ALL</property>          <property name="debugging">true</property>          <!-- Print log messages to console -->          <property name="logging.useParentHandlers">true</property>          <property name="logging.addConsoleHandler">java.util.logging.Level.ALL</property>      </properties>      <configurations>          <configuration name="standard" >              <capabilities>                  <initialcapability ref="ArrivalManagement" configuration="standard"/>              </capabilities>          </configuration>      </configurations>  </agent> |

1. ArrivalManagement.capability.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!--      <H3DCI4HD2D 3: AMAN capability.</H3>      Handle arrival requests optimizing DCI  -->  <capability xmlns="      xmlns:xsi="      xsi:schemaLocation="<http://jadex.sourceforge.net/jadex>        name="ArrivalManagement"      package="simulator.aman">        <imports>          <import>jadex.bridge.fipa.\*</import>          <import>java.lang.Long</import>          <import>simulator.environment.Flight</import>          <import>simulator.environment.CS3Flight</import>          <import>simulator.environment.CS3DCI4HD2DSimulation</import>      </imports>        <beliefs>          <beliefref name="simulationDCI4HD2D" >              <abstract/>          </beliefref>            <beliefref name="icao" >              <abstract/>          </beliefref>            <beliefsetref name="flights" >              <abstract/>          </beliefsetref>          </beliefs>        <plans>              <plan name="handleRequest" >              <body class="HandleDCIRequestsPlan"/>          </plan>        </plans>        <events>            <messageevent name="request\_CI\_priorities" direction="receive" type="fipa">              <parameter name="performative" class="String" direction="fixed">                  <value>request\_CI\_priorities</value>              </parameter>                  <match>$content instanceof Proposal;</match>         </messageevent>            <messageevent name="send\_CI\_priorities" direction="send" type="fipa">              <parameter name="performative" class="String" direction="fixed">                  <value>"send\_CI"</value>              </parameter>          </messageevent>        </events>        <expressions>          <expression name="get\_flight">              select one Flight $flight              from $beliefbase.flights              where $flight.getId() == $id          </expression>      </expressions>        <properties>          <!-- Output log -->          <property name="logging.level">java.util.logging.Level.ALL</property>          <property name="logging.level.exceptions">java.util.logging.Level.ALL</property>          <property name="debugging">true</property>          <!-- Print log messages to console -->          <property name="logging.useParentHandlers">true</property>          <property name="logging.addConsoleHandler">java.util.logging.Level.ALL</property>      </properties>        <configurations>          <configuration name="standard" >              <plans>                  <initialplan ref="handleRequest"/>              </plans>          </configuration>      </configurations>  </capability> |

1. Aircraft.java

|  |
| --- |
| package simulator.environment;      /\*\*   \* Aircraft is a subclass of Agent that extends its data about its type of aircraft   \*   \* @author Jorge Martin   \*/  public class Aircraft extends Agent {        /\*\* File path where the XML agent description is located \*/      private static final String PATH = "simulator/aircraft/Aircraft.agent.xml";        /\*\* Details of aircraft model \*/      private AircraftType subtype;        //++ Inherited attributes from abstract Agent Class        /\*\*       \* Sole constructor       \*/      public Aircraft() {          super();      }          /\*\*       \* Constructs an Aircraft object using all the attributes       \* @param id Icao identifier       \* @param type Aircraft model       \* @param path Aircraft XML agent description       \*/      public Aircraft(String id, AircraftType subtype) {          super(id, 8, PATH);          this.subtype = subtype;      }          /\*\*       \* @return the subtype       \*/      public AircraftType getSubtype() {          return subtype;      }          /\*\*       \* @param subtype the subtype to set       \*/      public void setSubtype(AircraftType subtype) {          this.subtype = subtype;      } |

1. AgentType.java

|  |
| --- |
| package simulator;  /\*\*   \* AgentType is an enum class to enumerate different types of agents   \*   \* @author Jorge Martin   \* @author IruneLansorena   \*   \*/  public enum AgentType {      /\*\* Airport class \*/      AIRPORT(0, 1),      /\*\* Airline class \*/      AIRLINE(0, 2),      /\*\* Full service airline \*/      FSC(3, 2),      /\*\* Regional airline \*/      REG(4,2),      /\*\* Low-cost airline\*/      LCC(5,2),      /\*\* Charter airline \*/      CHT(6,2),      /\*\* Unknown type airline \*/      XXX(7,2),      /\*\* AMAN class \*/      AMAN(0,3),      /\*\* DMAN class \*/      DMAN(0,4);        private final int subtype;      private final int type;        AgentType(int subtype, int type) {          this.type    = type;          this.subtype = subtype;      }        public static AgentType type(int type) {          AgentType result = null;          switch (type) {          case 1:              result = AIRPORT;              break;          case 2:              result = AIRLINE;              break;          case 3:              result = AMAN;              break;          case 4:              result = DMAN;              break;          default:              System.out.println("Agent type undefined");              break;          }          return result;      }        public static AgentType subtype(int id) {          AgentType result = null;          switch (id) {          case 3:              result = FSC;              break;          case 4:              result = REG;              break;          case 5:              result = LCC;              break;          case 6:              result = CHT;              break;          case 7:              result = XXX;              break;          default:              System.out.println("Agent subtype undefined");              break;          }          return result;      }        public int getType() { return type; }      public int getSubtype() { return subtype; }    } |

1. CS3DCI4HD2DSimulation.java

|  |
| --- |
| package simulator.environment;    import java.sql.ResultSet;  import java.sql.SQLException;  import java.util.ArrayList;  import java.util.Calendar;  import java.util.Date;  import java.util.GregorianCalendar;  import java.util.LinkedList;  import java.util.List;  import java.util.TimeZone;  import simulator.AgentType;  import simulator.SimulationType;  import simulator.db.DBAsyncAdapter;  import simulator.db.QueryT;  import simulator.db.SQLQuery;  import utils.Utils;    /\*\*   \* Class that extends Simulation to encapsulate additional simulation parameters for CS3Extension   \* @author IruneLansorena   \*   \*/  public class CS3DCI4HD2DSimulation extends Simulation {        private static final String GET\_CONNECTIONS = "SELECT \* FROM flight\_connection";      private static final String QUERY\_AGENT     = "call GET\_AGENT('%s')";      private static final String QUERY\_AIRCRAFT  = "call GET\_AIRCRAFT('%s')";      private static final String QUERY\_FLIGHTS   = "SELECT \* FROM flight";      private static final String QUERY\_FLIGHTSground\_improvements   = "SELECT \* FROM ground\_improvements";      private static final String QUERY\_FLIGHTSSESARground   = "SELECT \* FROM ground\_improvements";        /\*\* Start time when the data is collected \*\*/      private Date  starttime;        private float fuelprice;      private float co2ratio;      private String segment;      private String eval\_ap;      private boolean uncertainty;      //private SimulationType simulationType;      private List<Connection> connections;        public CS3DCI4HD2DSimulation() {            super();          System.out.println("Default CS3DCI4HD2DSimulation constructor");        }          //new parameter for 3 CS \* 3 different scenarios (reg)        public CS3DCI4HD2DSimulation (float fuelprice, float co2ratio, String segment, String eval\_ap, boolean uncertainty, CS3Regulation reg, int simulationType)              throws Exception {              //CASS extension id=4, each regulation to be defined by AB as well as parameters            super(4, reg, SimulationType.type(simulationType));            System.out.println("CS3DCI4HD2DSimulation constructor");          this.fuelprice   = fuelprice;          this.co2ratio    = co2ratio;          this.segment     = segment;          this.eval\_ap     = eval\_ap;          this.uncertainty = uncertainty;      }      //Need to change for adding new agents and other case study dependant attributes        @Override      protected void initializeSimulation() throws SQLException {          // DB Connector          DBAsyncAdapter cass\_db = DBAsyncAdapter.getInstance();          System.out.println("simu type "+ this.simulationType.getType());          // Query flights depends on what scenario is            switch (this.simulationType.getType()) {          //BASELINE              case 0:                  System.out.println("Baseline");                  ResultSet rs = (ResultSet) cass\_db.executeQuery(new SQLQuery(QUERY\_FLIGHTS, QueryT.EXECUTE));                  //we need a parameter to define with scenario is 0,1,2 in XML for BASELINE, GROUND OR SESAR+GROUND improvements.                  while (rs.next()) {                      String code = rs.getString(1);                      Airline carrier = (Airline) getDBAgent(rs.getString(2));                      AircraftType aircraft\_type = getDBAircraftType(rs.getString(3));                      String registration   = rs.getString(4);                      Airport dep\_airport = (Airport) getDBAgent(rs.getString(5));                      Airport arr\_airport = (Airport) getDBAgent(rs.getString(6));                        // Create one aircraft agent per IFPS                      Aircraft aircraft = null;                      if (!registration.equals("")) {                          aircraft = (Aircraft) agents.get(registration);                          if (aircraft == null) {                              aircraft = new Aircraft(registration, aircraft\_type);                              this.agents.put(registration, aircraft);                          }                      }                          long EOBT = rs.getTimestamp(7).getTime();                        long ETA = rs.getTimestamp(10).getTime();                          System.out.println("Checking TimeStamps UPM milliseconds EOBT "+ EOBT+ "ETA "+ ETA);                        int seats = rs.getInt(17);                        long SOBT = 0;                      if (rs.getTimestamp(18) != null) {                          SOBT = rs.getTimestamp(18).getTime();                      }                      long SIBT = 0;                      if (rs.getTimestamp(19) != null) {                          SIBT = rs.getTimestamp(19).getTime();                      }                        boolean fixed = rs.getBoolean(16);                          try {                          CS3Flight f = new CS3Flight(code, carrier, dep\_airport, arr\_airport, aircraft\_type,                                  EOBT, ETA, registration, seats, SOBT, SIBT, fixed, segment);                          this.flights.put(code, f);                            // Add flight to the agent's flight list                          carrier.addFlight(f);                          dep\_airport.addFlight(f);                          arr\_airport.addFlight(f);                          if (aircraft != null) {                              aircraft.addFlight(f);                          }                      }                      catch (Exception e) {                          System.out.println(e.getMessage());                      }                  }                      getDBConnections();                  calculateSimulationInterval();                  System.out.println("#Flights: " + flights.size());                break;            //GROUND          case 1:              ResultSet rsground = (ResultSet) cass\_db.executeQuery(new SQLQuery(QUERY\_FLIGHTSground\_improvements, QueryT.EXECUTE));              while (rsground.next()) {                  String code = rsground.getString(1);                    String GET\_BASELINE = "SELECT \* FROM FLIGHT WHERE FltNum = '"+ code+"' ";                  ResultSet rsgroundBaseline = (ResultSet) cass\_db.executeQuery(new SQLQuery(GET\_BASELINE, QueryT.EXECUTE));                  Airline carrier = (Airline) getDBAgent(rsgroundBaseline.getString(2));                  AircraftType aircraft\_type = getDBAircraftType(rsgroundBaseline.getString(3));                  String registration   = rsgroundBaseline.getString(4);                  Airport dep\_airport = (Airport) getDBAgent(rsgroundBaseline.getString(5));                  Airport arr\_airport = (Airport) getDBAgent(rsgroundBaseline.getString(6));                    // Create one aircraft agent per IFPS                  Aircraft aircraft = null;                  if (!registration.equals("")) {                      aircraft = (Aircraft) agents.get(registration);                      if (aircraft == null) {                          aircraft = new Aircraft(registration, aircraft\_type);                          this.agents.put(registration, aircraft);                      }                  }                    //some values changed for groundImprovements!                  long EOBT = 0;                  long ETA = 0;                    if (!Utils.isEmpty(rsground.getTimestamp("departureTime")) && !Utils.isEmpty(rsground.getTimestamp("arrivalTime"))){                      //departure and arrival changes due to connections                      EOBT = rsground.getTimestamp("departureTime").getTime();                      ETA =  rsground.getTimestamp("arrivalTime").getTime();                      //Check because of past gmt millseconds problems.                      System.out.println("Checking TimeStamps INNAXIS EOBT"+ rsground.getTimestamp("departureTime")+ "ETA "+ rsground.getTimestamp("arrivalTime"));                      System.out.println("Checking TimeStamps INNAXIS milliseconds EOBT"+ EOBT+ "ETA "+ ETA);                    }                  else {                  EOBT = rsgroundBaseline.getTimestamp(7).getTime();                    ETA = rsgroundBaseline.getTimestamp(10).getTime();                    System.out.println("Checking TimeStamps UPM milliseconds EOBT"+ EOBT+ "ETA "+ ETA);                  }                  int seats = rsgroundBaseline.getInt(17);                    long SOBT = 0;                  if (rsgroundBaseline.getTimestamp(18) != null) {                      SOBT = rsgroundBaseline.getTimestamp(18).getTime();                  }                  long SIBT = 0;                  if (rsgroundBaseline.getTimestamp(19) != null) {                      SIBT = rsgroundBaseline.getTimestamp(19).getTime();                  }                    boolean fixed = rsgroundBaseline.getBoolean(16);                      try {                      CS3Flight f = new CS3Flight(code, carrier, dep\_airport, arr\_airport, aircraft\_type,                              EOBT, ETA, registration, seats, SOBT, SIBT, fixed, segment);                      this.flights.put(code, f);                        // Add flight to the agent's flight list                      carrier.addFlight(f);                      dep\_airport.addFlight(f);                      arr\_airport.addFlight(f);                      if (aircraft != null) {                          aircraft.addFlight(f);                      }                  }                  catch (Exception e) {                      System.out.println(e.getMessage());                  }              }                break;          //GROUND+SESAR          case 2:  ...              break;            }        }      private void getDBConnections() throws SQLException{          DBAsyncAdapter a = DBAsyncAdapter.getInstance();          ResultSet rs = (ResultSet) a.executeQuery(new SQLQuery(GET\_CONNECTIONS, QueryT.EXECUTE));          this.connections = new LinkedList<Connection>();            while (rs.next()) {              int connectionid = rs.getInt(1);              String arr\_id   = rs.getString(2);              CS3Flight arr   = (CS3Flight) this.flights.get(arr\_id);              String dep\_id   = rs.getString(3);              CS3Flight dep   = (CS3Flight) this.flights.get(dep\_id);              int    mct      = rs.getInt(4) \* 60000;                if (arr != null && dep != null) {                  Connection c = new Connection(connectionid, arr, dep, mct);                  connections.add(c);              }          }      }          private void calculateSimulationInterval() {          for (Flight flight : this.flights.values()) {              CS3Flight f = (CS3Flight) flight;              if (starttime == null) {                  starttime = f.getEOBT();              }              else if (starttime.after(f.getEOBT())) {                  starttime = f.getEOBT();              }          }          Calendar c = new GregorianCalendar(TimeZone.getTimeZone("GMT"));          c.setTime(starttime);          c.add(Calendar.HOUR\_OF\_DAY, -2);            starttime = c.getTime();      }        private Agent getDBAgent(String icao) throws SQLException {          Agent agent = getSimAgent(icao);            if (agent == null) {              DBAsyncAdapter cass\_db = DBAsyncAdapter.getInstance();                SQLQuery query = new SQLQuery(String.format(QUERY\_AGENT, icao), QueryT.EXECUTE);              ResultSet selected\_agent = (ResultSet) cass\_db.executeQuery(query);                if (selected\_agent.first()) {                  // Create an agent with the following sort of attributes                  int subtype      = selected\_agent.getInt(2);                  String location  = selected\_agent.getString(4);                  double latitude  = selected\_agent.getDouble(5);                  double longitude = selected\_agent.getDouble(6);                    // TODO Extend agent classification                  if (subtype == 1) {                      agent = new Airport(icao,1,location,latitude,longitude);                  }                  else {                      agent = new Airline(icao,2, subtype);                  }                  agents.put(icao, agent);                    // Add the previous agent to the simulated list in the database                  String call = String.format(ADD\_SIMULATED\_AGENT, getId(), icao);                  cass\_db.executeQuery(new SQLQuery(call, QueryT.UPDATE));              }          }          return agent;      }        private AircraftType getDBAircraftType(String model) throws SQLException {            AircraftType result = getSimAircraftType(model);            if (result == null) {              DBAsyncAdapter cass = DBAsyncAdapter.getInstance();              SQLQuery query      = new SQLQuery(String.format(QUERY\_AIRCRAFT, model), QueryT.EXECUTE);                ResultSet rs = (ResultSet) cass.executeQuery(query);              if (rs.first()) {                  double mtow           = rs.getDouble(2);                  double co             = rs.getDouble(3);                  double nox            = rs.getDouble(4);                  double noise          = rs.getDouble(5);                  double avg\_seats      = rs.getDouble(6);                  double default\_ci     = rs.getDouble(8);                  double default\_usage  = rs.getDouble(9);                  char   wk\_tb          = rs.getString(12).charAt(0);                    result = new AircraftType(model, mtow, default\_ci, avg\_seats, co, nox, noise, default\_usage, wk\_tb);                  aircraft\_types.put(model, result);              }          }          return result;      }          /\*\*       \* Get the time interval of the simulated flights       \* @return An Interval element       \*/      public Date getStarttime() {          return this.starttime;      }          /\*\*       \* @param starttime the starttime to set       \*/      public void setStarttime(Date starttime) {          this.starttime = starttime;      }        /\*\*       \* @return the fuelprice       \*/      public float getFuelprice() {          return fuelprice;      }        /\*\*       \* @param fuelprice the fuelprice to set       \*/      public void setFuelprice(float fuelprice) {          this.fuelprice = fuelprice;      }        /\*\*       \* @return the co2ratio       \*/      public float getCo2ratio() {          return co2ratio;      }        /\*\*       \* @param co2ratio the co2ratio to set       \*/      public void setCo2ratio(float co2ratio) {          this.co2ratio = co2ratio;      }        /\*\*       \* @return the segment       \*/      public String getSegment() {          return segment;      }        /\*\*       \* @param segment the segment to set       \*/      public void setSegment(String segment) {          this.segment = segment;      }        /\*\*       \* @return the eval\_ap       \*/      public String getEval\_ap() {          return eval\_ap;      }        /\*\*       \* @param eval\_ap the eval\_ap to set       \*/      public void setEval\_ap(String eval\_ap) {          this.eval\_ap = eval\_ap;      }          /\*\*       \* @return the uncertainty       \*/      public boolean isUncertainty() {          return uncertainty;      }      /\*\*       \* @param uncertainty the uncertainty to set       \*/      public void setUncertainty(boolean uncertainty) {          this.uncertainty = uncertainty;      }      public List<Connection> getConnections() {          return this.connections;      }          /\*\*       \* @param connections the connections to set       \*/      public void setConnections(List<Connection> connections) {          this.connections = connections;      }          public List<Connection> getArrConnections(CS3Flight flight) {          List<Connection> result = new ArrayList<Connection>();          for(Connection c : connections) {              if (flight.equals(c.getArrival())) {                  result.add(c);              }          }          return result;      }          public List<Connection> getDepConnections(CS3Flight flight) {          List<Connection> result = new ArrayList<Connection>();          for(Connection c : connections) {              if (flight.equals(c.getDeparture())){                  result.add(c);              }          }          return result;      }          public void addConnection(Connection c) {          this.connections.add(c);      }          public boolean removeConnection(Connection c) {          return this.connections.remove(c);      }        public int calculateMTT(CS3Flight current) {          // TODO Extend turnaround calculation (40 min)          int time = 2400000;          AircraftType model = current.getAircraftType();          Airport dep\_ap  = current.getDep\_airport();          Airline carrier = current.getCarrier();          AgentType ao\_type = AgentType.subtype(carrier.getSubtype());          switch(ao\_type) {          case FSC:                if (dep\_ap.getId().equals(getEval\_ap())) {                    if (carrier.getId().equals("SWR")){                      if (model.getWkTBL() == 'H') {                          time = 90 \* 60000;                      }                      else if (model.getWkTBL() == 'M') {                          time = 23 \* 60000;                      }                  }                  else{                      if (model.getWkTBL() == 'J') {                          time = 180 \* 60000;                      }                      else if (model.getWkTBL() == 'H') {                          time = 102 \* 60000;                      }                      else if (model.getWkTBL() == 'M') {                          time = 22 \* 60000;                      }                  }              }              else {                  if (model.getWkTBL() == 'H') {                      time = 96 \* 60000;                  }                  else if (model.getWkTBL() == 'M') {                      time = 22 \* 60000;                  }              }              break;          case LCC:              if (dep\_ap.getId().equals(getEval\_ap())) {                  if (model.getWkTBL() == 'H') {                      time = 80 \* 60000;                  }                  else if (model.getWkTBL() == 'M') {                      time = 26 \* 60000;                  }              }              else {                  if (model.getWkTBL() == 'H') {                      time = 80 \* 60000;                  }                  else if (model.getWkTBL() == 'M') {                      time = 22 \* 60000;                  }              }              break;          case REG:              if (dep\_ap.getId().equals(getEval\_ap())) {                  time = 17 \* 60000;              }              else {                  if (model.getWkTBL() == 'H') {                      time = 80 \* 60000;                  }                  else if (model.getWkTBL() == 'M') {                      time = 18 \* 60000;                  }              }              break;          case CHT:              if (dep\_ap.getId().equals(getEval\_ap())) {                  time = 37 \* 60000;              }              else {                  if (model.getWkTBL() == 'H') {                      time = 80 \* 60000;                  }                  else if (model.getWkTBL() == 'M') {                      time = 40 \* 60000;                  }              }              break;          default:              break;          }          return time;      }  } |

1. Simulation.java

|  |
| --- |
| package simulator.environment;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.util.ArrayList;  import java.util.Collection;  import java.util.HashMap;  import java.util.List;  import java.util.Map;  import simulator.SimulationType;  import simulator.db.DBAsyncAdapter;  import simulator.db.QueryT;  import simulator.db.SQLQuery;  /\*\*   \* Represents a simulation performed in the CASSIOPEIA simulation tool   \* It must be extended because of the singularities of the flights in each case study   \*   \* @author Jorge MartÃ­n   \* @author IruneLansorena   \* @version 3.0   \* Changes: add simulationType for baseline, ground, SESAR+ground   \* @version 2.0   \* Changes: 2.0. Redefine simulation object to manage the simulation input data   \*          1.0. Initial version   \*/  public abstract class Simulation {      public static final String DATETIME\_FORMAT = "yyyy-MM-dd HH:mm:ss";        //Adding simulation type to the NEW\_SIMULATION procedure.      private static final String ADD\_SIMULATION       = "CALL NEW\_SIMULATION('%d', '%d','%d')";      protected static final String ADD\_SIMULATED\_AGENT  = "CALL NEW\_SIMULATED\_AGENT('%d', '%s')";        /\*\* The number which the simulation is saved in the database \*\*/      private int        id;      /\*\* A number to identify which case study are simulating \*\*/      private int        case\_study;      /\*\* Regulations that is going to be applied \*\*/      private Regulation regulation;      /\*\* SimulationType \*\*/      protected SimulationType simulationType;      /\*\* Agents used in the simulation \*\*/        protected Map<String, Agent>    agents;      protected Map<String, Flight>   flights;      protected Map<String, AircraftType> aircraft\_types;            public Simulation() {}        /\*\*       \* Creates a new object providing the required arguments       \* @param case\_study Case study id, for visualization purposes       \* @param reg Regulation object created or obtained from the database       \* @param simulationType in order to work with baseline, ground or ground+sesar scenario configurations.       \* @throws SQLException When creating the simulation in the database       \*/          public Simulation(int case\_study, Regulation reg, SimulationType simulationType) throws Exception {          System.out.println("Super" ); //warning!          // Set the following attributes          this.case\_study     = case\_study;          this.regulation     = reg;          this.agents         = new HashMap<String, Agent>();          this.aircraft\_types = new HashMap<String, AircraftType>();          this.flights        = new HashMap<String, Flight>();          this.simulationType = simulationType;          // Save simulation on database and get simulated flights          newSimulation();      }        /\*       \* Private method for create a simulation instance in the database       \*/      private void newSimulation() throws SQLException      {            System.out.println("Regulation before database "+regulation.getId());          System.out.println("Case Study before database "+case\_study);          //insert to the DB simuType needed!!!          int regid = 0;          if (regulation != null) {              regid = regulation.getId();          }          String query = String.format(ADD\_SIMULATION, regid, case\_study,id);          DBAsyncAdapter a = DBAsyncAdapter.getInstance();          System.out.println("Query "+ query);          ResultSet rs = (ResultSet) a.executeQuery(new SQLQuery(query, QueryT.EXECUTE));            if (rs.first()) {              this.id = rs.getInt(1);          }          initializeSimulation();      }        /\*\*       \* Abstract method to obtain the agents for the simulation, using the query       \* contained in the sql parameter       \*/      protected abstract void initializeSimulation() throws SQLException;            /\* GET METHODS FOR SIMPLE ATTRIBUTES \*/        ...        /\*\*       \* Private method used to get an aircraft instance from the database using       \* the model parameter       \*       \* @parameter model The model name of the aircraft       \*/      public AircraftType getSimAircraftType(String model) throws SQLException {          return this.aircraft\_types.get(model);      }          public Flight getSimFlight(String id) {          return this.flights.get(id);      }          public Collection<Flight> getSimFlights(){          return this.flights.values();      }          /\*\*       \* It insert the simulation output into the database, indicating the time point       \*       \* @param time\_point Could be an step (discrete) or a time value (long value)       \* @param element An element who implements the persistence interface       \* @return An String representation of the inserted element       \*/      public String insertSimulationResult(long time\_point, Persistence element) {          DBAsyncAdapter cass\_db = DBAsyncAdapter.getInstance();            String sql = element.saveStatement(id, time\_point);          SQLQuery query = new SQLQuery(sql, QueryT.EXECUTE);          cass\_db.executeAsyncQuery(query);          return element.toString(time\_point);      }  } |

1. Manager.agent.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!--      <H3>Manager: Case 3Extension.</H3>      Behavior for case 3Extension  -->  <agent xmlns="      xmlns:xsi="      xsi:schemaLocation="http://jadex.sourceforge.net/jadex                          http://jadex.sourceforge.net/jadex-bdi-2.3.xsd"      name="Manager"      package="simulator.manager">        <imports>          <import>jadex.bridge.fipa.\*</import>          <import>simulator.environment.Simulation</import>          <import>simulator.environment.Flight</import>          <import>simulator.environment.CS3Flight</import>          <import>simulator.environment.CS3DCI4HD2DSimulation</import>          <import>simulator.environment.Agent</import>          <import>simulator.environment.Airline</import>          <import>simulator.environment.Airport</import>          <import>simulator.environment.Aircraft</import>      </imports>        <capabilities>          <!-- Capability to start other agents. -->          <capability name="cmscap" file="jadex.bdi.planlib.cms.CMS"/>      </capabilities>        <beliefs>          <belief name="simulationDCI4HD2D" class="Simulation" />          <beliefset name="flights"   class="Flight" />            <beliefset name="aircrafts" class="Aircraft" />          <beliefset name="airlines" class="Airline" />          <beliefset name="airports" class="Airport" />      </beliefs>        <goals>          <!-- Used to start other agents. -->          <achievegoalref name="cms\_create\_component">              <concrete ref="cmscap.cms\_create\_component"/>          </achievegoalref>            <performgoal name="create\_agents">              <parameterset name="agents"  class="Agent" direction="in"/>          </performgoal>            <performgoal name="create\_indicators" />      </goals>        <plans>          <plan name="protocol"  >              <body class="ManagerProtocolPlan" />              <waitqueue>                  <messageevent ref="confirm\_simulation"/>                  <messageevent ref="inform\_allocated"/>              </waitqueue>          </plan>            <plan name="create\_agents" >              <body class="CreateAgentsPlan" />              <trigger>                  <goal ref="create\_agents" />              </trigger>              <waitqueue>                  <messageevent ref="confirm\_initialized"></messageevent>              </waitqueue>          </plan>          <plan name="create\_indicators" >              <body class="CreateIndicatorsPlan" />              <trigger>                  <goal ref="create\_indicators"/>              </trigger>          </plan>      </plans>        <events>          <messageevent name="confirm\_initialized" direction="receive" type="fipa">              <parameter name="performative" class="String" direction="fixed" >                  <value>"confirm\_initialized"</value>              </parameter>          </messageevent>          <messageevent name="perform\_simulation" direction="send" type="fipa">              <parameter name="performative" class="String" direction="fixed" >                  <value>"perform\_simulation"</value>              </parameter>          </messageevent>          <messageevent name="confirm\_simulation" direction="receive" type="fipa">              <parameter name="performative" class="String" direction="fixed" >                  <value>"confirm\_simulation"</value>              </parameter>          </messageevent>            <messageevent name="request\_allocation\_costs" direction="send" type="fipa">              <parameter name="performative" class="String" direction="fixed" >                  <value>"request\_allocation\_costs"</value>              </parameter>          </messageevent>          <!-- Message received by the manager to generate the indicators -->          <messageevent name="inform\_allocated" direction="receive" type="fipa" >              <parameter name="performative" class="String" direction="fixed" >                  <value>"inform\_allocated"</value>              </parameter>          </messageevent>      </events>      <expressions>          <expression name="query\_airport\_flights">              select CS3Flight $flight              from $beliefbase.flights              where $flight.getDep\_airport().getId().equals($id) ||                    $flight.getArr\_airport().getId().equals($id)          </expression>            <expression name="query\_airline\_flights">              select CS3Flight $flight              from $beliefbase.flights              where $flight.getCarrier().getId().equals($id)          </expression>            <expression name="query\_flights">              select CS3Flight $flight              from $beliefbase.flights              order by flight.getIFPS()          </expression>            <expression name="query\_flight" >              select CS3Flight $flight              from $beliefbase.flights              where $flight.getId() == $id          </expression>      </expressions>      <properties>          <!-- Output log -->          <property name="logging.level">java.util.logging.Level.ALL</property>          <property name="logging.level.exceptions">java.util.logging.Level.ALL</property>          <property name="debugging">true</property>          <!-- Print log messages to console -->          <property name="logging.useParentHandlers">true</property>          <property name="logging.addConsoleHandler">java.util.logging.Level.ALL</property>      </properties>        <configurations>          <configuration name="standard">              <plans>                  <initialplan ref="protocol" />              </plans>          </configuration>      </configurations>    </agent> |

1. ManagerProtocolPlan.java

|  |
| --- |
| package simulator.manager;    import java.util.Collection;  import java.util.Map;  import simulator.db.DBAsyncAdapter;  import simulator.environment.Aircraft;  import simulator.environment.Airline;    ...  import jadex.bridge.fipa.SFipa;  import jadex.commons.future.DefaultResultListener;  import jadex.commons.future.IFuture;  /\*\*   \* Master Plan of the Simulation   \* @author IruneLansorena   \*/  public class ManagerProtocolPlan extends Plan {      private static final long serialVersionUID = 1L;        /\*\* Simulation object \*/      private CS3DCI4HD2DSimulation sim;        @Override      public void body() {          System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* START: " + System.currentTimeMillis() +                  " \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");          try {              // Ask for GUI parameters              IExternalAccess app = getScope().getParentAccess();              IFuture<Map<String, Object>> fArguments = app.getArguments();              fArguments.addResultListener(new DefaultResultListener<Map<String, Object>>(){                  @Override                  public void resultAvailable(Map<String, Object> result) {                      initialization(result);                  }              });                // Save simulated flights              Collection<Flight> flights = sim.getSimFlights();              getBeliefbase().getBeliefSet("flights").addFacts(flights.toArray());                // Create agents in the simulation platform              IGoal g1 = createGoal("create\_agents");              g1.getParameterSet("agents").addValues(sim.getSimAgents().toArray());              dispatchSubgoalAndWait(g1);                System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* START SIMULATION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");                // Perform simulation              Aircraft[] aircrafts = (Aircraft[]) getBeliefbase().getBeliefSet("aircrafts").getFacts();              System.out.println("Loaded Aircrafts");              for (Aircraft aircraft : aircrafts) {                  IMessageEvent out = createMessageEvent("perform\_simulation");                  IComponentIdentifier cid = new ComponentIdentifier(aircraft.getId(), getScope().getComponentIdentifier().getParent());                  out.getParameterSet(SFipa.RECEIVERS).addValue(cid);                  sendMessage(out);              }              // Wait for all aircraft simulation              for (int i = 0; i < aircrafts.length; i++) {                  waitForMessageEvent("confirm\_simulation");              }              System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* RESULTS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");                // Allocation of delay costs for each airline              Airline[] airlines = (Airline[]) getBeliefbase().getBeliefSet("airlines").getFacts();              for (Airline airline : airlines) {                  IMessageEvent out = createMessageEvent("request\_allocation\_costs");                  IComponentIdentifier cid = new ComponentIdentifier(airline.getId(), getScope().getComponentIdentifier().getParent());                  out.getParameterSet(SFipa.RECEIVERS).addValue(cid);                  sendMessage(out);              }              // Wait for delay cost allocation              for (int i = 0; i < airlines.length; i++) {                  waitForMessageEvent("inform\_allocated");              }                System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* INDICATORS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");              // Create indicators              IGoal g4 = createGoal("create\_indicators");              dispatchSubgoalAndWait(g4);                // Wait for insert all values to the database              DBAsyncAdapter.getInstance().waitAndDisconnect();                System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* END: " + System.currentTimeMillis() +                                                                  " \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");          }          catch (Exception e) {              e.printStackTrace();          }          killAgent();      }        /\*\*       \* Initialize simulation using parameters from GUI       \* @param result values from GUI       \*/      private void initialization(Map<String, Object> result) {          // Load simulation parameters          float fuelprice = (float) result.get("fuelprice");          float co2ratio  = (float) result.get("co2ratio");          String segmentid = (String) result.get("segmentid");          String airportid = (String) result.get("airportid");          boolean uncertainty = (boolean) result.get("uncertainty");          int regulationid = (int)    result.get("regulationid");          int simulationType = (int) result.get("simulationType");              try {              // Create new simulation and save it (specified on DCI4HD2D.application.xml and selected from GUI, 1 default scenario (baseline UPM))              CS3Regulation reg  = new CS3Regulation(regulationid);              this.sim = new CS3DCI4HD2DSimulation (fuelprice, co2ratio, segmentid, airportid, uncertainty, reg, simulationType);              //setting the fact of concrete simulation. In this case is the simulation of CASS++              System.out.println("Setting the fact of concrete simulation");              getBeliefbase().getBelief("simulationDCI4HD2D").setFact(sim);          }          catch (Exception e) {              e.printStackTrace();          }      }  } |

1. CreateAgentsPlan.java

|  |
| --- |
| package simulator.manager;    import java.util.HashMap;  import java.util.Map;  import simulator.environment.AMAN;  import simulator.environment.Agent;  import simulator.environment.Aircraft;  import simulator.environment.Airline;  import simulator.environment.Airport;  import simulator.environment.CS3DCI4HD2DSimulation;  import simulator.environment.DMAN;  import jadex.bdi.runtime.GoalFailureException;  import jadex.bdi.runtime.IGoal;  import jadex.bdi.runtime.Plan;  /\*\*   \* Plan to create agents in Jadex Platform   \* @author Jorge Martin   \* @author IruneLansorena CS3Extension (DCI4HD2D)   \*   \*/  public class CreateAgentsPlan extends Plan{      private static final long serialVersionUID = 1L;        /\*\* Simulation object for CASS++ \*/      private CS3DCI4HD2DSimulation sim;        /\*\*       \* Default constructor to get Simulation belief       \*/      public CreateAgentsPlan() {          this.sim = (CS3DCI4HD2DSimulation) getBeliefbase().getBelief("simulationDCI4HD2D").getFact();      }      @Override      public void body() {            System.out.println("Creating Agents Body");          // Get simulated agents          IGoal g = (IGoal) getReason();          Agent[] agents = (Agent[]) g.getParameterSet("agents").getValues();            // For each agent, create a goal to create the agent          int created = 0;          for(Agent a : agents) {              // Create goal              final IGoal start = getGoalbase().createGoal("cms\_create\_component");              // Add goal parameters              start.getParameter("type").setValue(a.getPath());              start.getParameter("name").setValue(a.getId());              start.getParameter("parent").setValue(getComponentIdentifier().getParent());              start.getParameter("configuration").setValue("standard");                // Provide instance arguments              Map<String, Object> args = new HashMap<String, Object>();              args.put("simulation", sim);              args.put("icao",       a.getId());              args.put("flights",    a.getFlights());                // Add to the list of agents              if (a instanceof Airport) {                  Airport airport = (Airport) a;                  getBeliefbase().getBeliefSet("airports").addFact(airport);              }              else if (a instanceof Airline) {                  Airline airline = (Airline) a;                  args.put("type",  airline.getSubtype());                  getBeliefbase().getBeliefSet("airlines").addFact(airline);              }              else if (a instanceof Aircraft) {                  Aircraft aircraft = (Aircraft) a;                  args.put("model", aircraft.getSubtype());                  getBeliefbase().getBeliefSet("aircrafts").addFact(aircraft);              }              //introducing AMAN and DMAN              else if (a instanceof AMAN) {                  AMAN aman = (AMAN) a;                  getBeliefbase().getBeliefSet("aman").addFact(aman);              }              else if (a instanceof DMAN) {                  DMAN dman = (DMAN) a;                  getBeliefbase().getBeliefSet("dman").addFact(dman);              }              start.getParameter("arguments").setValue(args);                // Dispatch goal to create the agent              try {                  dispatchSubgoalAndWait(start);                  created++;              }              catch (GoalFailureException e) {                  System.err.println(a.getId() + " agent creation fails");              }          }          System.out.println("#Agents:  " + created);            // Wait for agent confirmation          for (int i = 1; i < created ; i++) {              waitForMessageEvent("confirm\_initialized");          }      }  } |

1. MessageType.java

|  |
| --- |
| package simulator;  /\*\*   \* Enum type of messages   \* @author Jorge Martin   \* @author IruneLansorena   \*   \*/  public enum MessageType {      /\*\* Request airport delay \*/      REQUEST\_DELAY (0, "Request delay"),      /\*\* Inform airport delay \*/      INFORM\_DELAY(1, "Inform delay"),      /\*\* Request wait for pax time to the airline \*/      REQUEST\_WAIT\_TIME(2, "Request wait time"),      /\*\* Inform wait for pax time to the aircraft \*/      INFORM\_WAIT\_TIME(3, "Inform wait time"),      /\*\* Request ci change to the airline \*/      REQUEST\_CI(4, "Request CI"),      /\*\* Inform ci change to the aircraft \*/      INFORM\_CI(5, "Inform CI"),      /\*\* Request arrival slot to AMAN agent\*/      REQUEST\_ARRVSLOT(6,"Request departure slot"),      /\*\* Request departure slot to AMAN agent\*/      REQUEST\_DEPSLOT(7,"Request departure slot");            private final int    code;      private final String desc;        MessageType(int code, String desc) {          this.code    = code;          this.desc    = desc;      }        public int    code()    { return code; }      public String desc()    { return desc; }          public static MessageType value(int code) {          MessageType result = null;          switch (code) {          case 0:              result = REQUEST\_DELAY;              break;          case 1:              result = INFORM\_DELAY;              break;          case 2:              result = REQUEST\_WAIT\_TIME;              break;          case 3:              result = INFORM\_WAIT\_TIME;              break;          case 4:              result = REQUEST\_CI;              break;          case 5:              result = INFORM\_CI;              break;            case 6:              result = REQUEST\_ARRVSLOT;              break;            case 7:              result = REQUEST\_DEPSLOT;              break;            default:              System.out.println("Message type undefined");              break;          }          return result;      }  } |

1. RequestArrivSlot.java

|  |
| --- |
| package simulator;  import java.util.Date;  import java.util.List;  import simulator.environment.Message;  import simulator.environment.Slot;  /\*\*   \* Message encapsulating FIPA compliant specifications for approach slot request   \* (Aicraft-->AMAN)   \*   \*/  public class RequestArrivSlot extends Message {      private String flightid;      private Date Tci;      private Date Tmin;      private Date Tmax;      private List<Slot> potentialslots;      /\*\*       \*       \*/      public RequestArrivSlot() {          super();      }        /\*\*       \* @param sender       \* @param receiver       \* @param type       \* @param desc       \*/      public RequestArrivSlot(String sender, String receiver, String flightid, Date Tci, Date Tmin, Date Tmax) {          super(sender, receiver, MessageType.REQUEST\_CI.code(), MessageType.REQUEST\_CI.desc());          this.flightid = flightid;          this.Tci = Tci;          this.Tmin = Tmin;          this.Tmax = Tmax;      }        @Override      public String content() {          String result= "Flight: " + flightid + " AMAN reply with potential slots:";          for (int i=0 ; i <= potentialslots.size();){              result+= result + "slot"+i+ potentialslots.get(i).toString();          }          return result;      }      /\*\*       \* @return the flightid       \*/      public String getFlightid() {          return flightid;      }      /\*\*       \* @param flightid the flightid to set       \*/      public void setFlightid(String flightid) {          this.flightid = flightid;      }        /\*\*       \* @return the tci       \*/      public Date getTci() {          return Tci;      }        /\*\*       \* @param tci the tci to set       \*/      public void setTci(Date tci) {          Tci = tci;      }        /\*\*       \* @return the tmin       \*/      public Date getTmin() {          return Tmin;      }        /\*\*       \* @param tmin the tmin to set       \*/      public void setTmin(Date tmin) {          Tmin = tmin;      }        /\*\*       \* @return the tmax       \*/      public Date getTmax() {          return Tmax;      }        /\*\*       \* @param tmax the tmax to set       \*/      public void setTmax(Date tmax) {          Tmax = tmax;      }        /\*\*       \* @return the potentialslots       \*/      public List<Slot> getPotentialslots() {          return potentialslots;      }        /\*\*       \* @param potentialslots the potentialslots to set       \*/      public void setPotentialslots(List<Slot> potentialslots) {          this.potentialslots = potentialslots;      }      } |
| /\*\*   \*   \*/  package simulator;  /\*\*   \* @author IruneLansorena   \* Enabler class to model different scenarios for sensitivity analysis   \*   \*/  public class SimuType {        public enum SimulationType {          /\*\* BASELINE \*/          BASELINE,          /\*\* SESAR IMPROVEMENTS \*/          SESAR,          /\*\* GROUND IMPROVEMENTS \*/          GROUND,          /\*\* GROUND+SESAR IMPROVEMENTS \*/          GROUNDSESAR      }        private SimulationType simutype;      /\*\*       \* @return the simutype       \*/      public SimulationType getSimutype() {          return simutype;      }      /\*\*       \* @param simutype the simutype to set       \*/      public void setSimutype(SimulationType simutype) {          this.simutype = simutype;      }    } | |

1. IndicatorType.java

|  |
| --- |
| package simulator;  /\*\*   \* Enum Type of Indicators   \*   \* @author IruneLansorena   \* @author Jorge Martin   \*   \*/  public enum IndicatorType {      /\*\* Percentage of flights departing on time - OB\_delay2 \*/      P1("Percentage of flights departing on time"), // Field TO\_delay      /\*\* Percentage of flights arriving on time - Arr\_delay2 \*/      P2("Percentage of flights arriving on time"),  // Field Arrival\_delay      /\*\* Standard deviation of delay of arrival times - Arr\_delay2 \*/      P3("Standard deviation of delay of arrival times"), //Field Arrival\_delay      /\*\* Total arrival delay - Arr\_delay2 \*/      P4("Total arrival delay"), // Field arrival\_delay      /\*\* Total delay costs - OwnDelay - getActualFc2  \*/      C1("Delay costs"), //Various fields (next flight costs + non\_connecting costs + connecting costs)      /\*\* Total fuel costs - getActualfc2 \*/      C2("Fuel costs"),  // Field fuel costs      /\*\* Total net costs - C1 + C2 \*/      C3("Net costs"), // C1 + C2      /\*\* Total CO2 emissions - getActualFc2 / fuel\_unit \* co2 unit \*/      E1("Environmental CO2 emissions"),// C2 \* co2 unit      /\*\* Gate-to-gate passenger trip time \*/      G2G("Gate-to-gate passenger trip time "), //minutes      /\*\* Door-to-door passenger trip time \*/      D2D ("Door-to-door passenger trip time"), //minutes      C4("Social cost"), //euros      MC("Missed connections"), //number of passengers      C5 ("Airlines costs"), //euros      C6("Hub airline cost"),//euros      C7("Hub airline deviation from optimal cost"), //euros      C8("Non-hub airline cost"), //      /\*\* Complexity of the solution\*/      P5("Speed variations"),      M1("Exchanged messages"), //number of messages      PAXD("PAX delay"); //minutes      private String desc;      IndicatorType(String desc) {          this.desc = desc;      }      public String getDescription() {          return desc;      }      public void setDescription(String desc) {          this.desc = desc;      }  } |

1. CreateIndicatorsPlan.java

|  |
| --- |
| package simulator.manager;  import org.apache.commons.math3.stat.descriptive.SummaryStatistics;  import simulator.ATMPlan;  import simulator.IndicatorType;  import simulator.environment.CS3Flight;  import simulator.environment.Flight;  import simulator.environment.Indicator;  /\*\*   \* ATM Plan to create indicators for the simulation   \*   \* @author Jorge Martin   \* @author IruneLansorena   \* @version 2.0   \* Changes: DCI4HD2DIndicators added   \* @see    simulator.ATMPlan   \*/  public class CreateIndicatorsPlan extends ATMPlan{      private static final long serialVersionUID = -5357766919434462733L;      @Override      public void body() {          // Get all simulation flights and initialize adders          Flight[] flights = (Flight[]) getBeliefbase().getBeliefSet("flights").getFacts();            SummaryStatistics stats = new SummaryStatistics();            double dep\_ontime\_3 = 0;          double dep\_ontime\_10 = 0;          double dep\_ontime\_15 = 0;            double arr\_ontime\_3 = 0;          double arr\_ontime\_10 = 0;          double arr\_ontime\_15 = 0;            double fuel\_costs = 0;          double delay\_costs = 0;            // For each flight          for (Flight aux : flights) {              CS3Flight flight = (CS3Flight) aux;                double arr\_delay = flight.getArrDelay2()/60000f;              double dep\_delay = flight.getOBdelay2()/60000f;                // Descriptive analysis              if (arr\_delay > 0) {                  stats.addValue(arr\_delay);              }                // Arrival delays              if (arr\_delay < 3) {                  arr\_ontime\_3++;              }              if (arr\_delay < 10) {                  arr\_ontime\_10++;              }              if (arr\_delay < 15) {                  arr\_ontime\_15++;              }                // Departure delays              if (dep\_delay < 3) {                  dep\_ontime\_3++;              }              if (dep\_delay < 10) {                  dep\_ontime\_10++;              }              if (dep\_delay < 15) {                  dep\_ontime\_15++;              }                // Delay costs              fuel\_costs  += flight.getActual\_fc2();              delay\_costs += flight.OwnDelayCosts() - flight.getActual\_fc2();          }            //Generating new indicators          // Indicator P1 Departure delay          IndicatorType t1 = IndicatorType.P1;          Indicator i1\_3  = new Indicator(t1.ordinal(), t1.getDescription(),  "3", dep\_ontime\_3/flights.length);          Indicator i1\_10 = new Indicator(t1.ordinal(), t1.getDescription(), "10", dep\_ontime\_10/flights.length);          Indicator i1\_15 = new Indicator(t1.ordinal(), t1.getDescription(), "15", dep\_ontime\_15/flights.length);            ...            // Indicator C2 Total fuel costs          IndicatorType t6 = IndicatorType.C2;          Indicator i6  = new Indicator(t6.ordinal(), t6.getDescription(),  null, fuel\_costs);            // Indicator C3 Total net costs          IndicatorType t7 = IndicatorType.C3;          Indicator i7  = new Indicator(t7.ordinal(), t7.getDescription(),  null , fuel\_costs + delay\_costs);            // Indicator EI Total CO2 emissions          IndicatorType t8 = IndicatorType.E1;          Indicator i8  = new Indicator(t8.ordinal(), t8.getDescription(),  null, fuel\_costs / sim.getFuelprice() \* sim.getCo2ratio());          // Add indicators to the database (storage for later analysis).          sim.insertSimulationResult(getTime(), i1\_3);          ....      }  } |

1. Pre-processing: flight plans inbound and outbound data manipulation and ingestion.

|  |
| --- |
| package simulator.db;  import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.sql.Time;  import java.util.ArrayList;  import java.util.Date;  import java.util.List;  import java.util.Map.Entry;  import java.util.TreeMap;  import java.util.concurrent.TimeUnit;    import simulator.environment.CS3SESARImprovedAircraft;  import simulator.environment.CS3SESARImprovedFlight;  import simulator.environment.Segment;  import utils.DateHandler;  import utils.ExcelHandler;  import utils.Haversine;  import utils.Utils;  /\*\*   \* @author IruneLansorena   \*  Data Pre-processing Functionality   \*  Database accessing and mapping between relational model and object oriented model   \*/  public class SESARImprovements {        public final static String connectionUrl = "jdbc:mysql://localhost/cassiopeia\_cs3";      private static final String host = connectionUrl;      private static final String user = "cass";      private static final String dbPwd  = "cassiopeia";      private List <CS3SESARImprovedFlight> improvedFlights = new ArrayList <CS3SESARImprovedFlight>();        public static Connection conn=null;            // method to create MySQL connections (Singleton Pattern)          public static Connection createConnection() {              //Connection conn=null;              // Use DRIVER and DBURL to create a connection              // Recommend connection pool implementation/usage              if (conn==null){              try {                    conn= DriverManager.getConnection(host, user, dbPwd);              } catch (SQLException e) {                  System.out.println(e.getMessage());              }              }              return conn;          }            public static void closeConnection() {          // Use DRIVER and DBURL to create a connection          // Recommend connection pool implementation/usage          try {              conn.close();          } catch (SQLException e) {             System.out.println(e.getMessage());          }        }    public void trajectoryImprovements () throws SQLException{            Connection conn = SESARImprovements.createConnection();          ResultSet rssegments = null;          ResultSet rsflights = null;          ResultSet dep = null;          ResultSet arriv = null;          List <CS3SESARImprovedFlight> improvedFlights = new ArrayList <CS3SESARImprovedFlight>();          String selectFlights= "SELECT \* FROM cassiopeia\_cs3.flight";          rsflights = conn.createStatement().executeQuery(selectFlights);              while (rsflights.next()){                String selectSegments= "SELECT \* FROM cassiopeia\_cs3.flight\_segment where FltNum = '"+ rsflights.getString(1)+"' order by sequence";                String selectGeoLocationDepart = "SELECT latitude,longitude FROM agent WHERE idagent = '"+rsflights.getString(5)+"'";                String selectGeoLocationArriv = "SELECT latitude,longitude FROM agent WHERE idagent = '"+rsflights.getString(6)+"'";                  CS3SESARImprovedFlight flight = new CS3SESARImprovedFlight ();              flight.setADEP(rsflights.getString(5));                flight.setADES(rsflights.getString(6));                dep = conn.createStatement().executeQuery(selectGeoLocationDepart);              arriv = conn.createStatement().executeQuery(selectGeoLocationArriv);              double latitudeDEP = 0.0;              double longitudeDEP = 0.0;              double latitudeArriv = 0.0;              double longitudeArriv = 0.0;                if (dep.next()){                  latitudeDEP = dep.getDouble(1);                  longitudeDEP= dep.getDouble(2);              }              if (arriv.next()){                  latitudeArriv= arriv.getDouble(1);                  longitudeArriv = arriv.getDouble(2);              }                flight.setFltNum(rsflights.getString(1));              improvedFlights.add(flight);              int maxFL=0;              int maxFLtotal= 0;              rssegments = conn.createStatement().executeQuery(selectSegments);              while (rssegments.next()) {                  if (rssegments.isFirst()){                      latitudeDEP = rssegments.getDouble(6);                      longitudeDEP = rssegments.getDouble(7);                  }                  if (rssegments.isLast()){                      latitudeArriv = rssegments.getDouble(6);                      longitudeArriv = rssegments.getDouble(7);                  }                  if (maxFLtotal<=rssegments.getInt(3) ){                      maxFLtotal =  rssegments.getInt(3);                  }                  //get MaxFL on cruise                  if (rssegments.getInt(5) == 2 && maxFL<=rssegments.getInt(3) ){                      maxFL=rssegments.getInt(3);                  }              }                //change from 28.12.2014 change is not based on airports, last points instead              flight.setMaxFL(maxFL);              flight.setMaxFLTotal(maxFLtotal);              flight.setAirportsDistance(Haversine.distFrom(latitudeDEP, longitudeDEP, latitudeArriv, longitudeArriv));                }            for (CS3SESARImprovedFlight flight: improvedFlights){                System.out.println("Nuevo vuelo" );              boolean found= false;              boolean found2= false;              double distanceClimb =0.0;              double distanceDescent =0.0;              double distanceCruise =0.0;              long distanceTimeClimb = 0;              long distanceTimeDescent = 0;              long distanceTimeCruise = 0;              double groundSpeed = 0.0;              double cruiseTime = 0.0;              long diffSeconds = 0;              double cruiseDistance = 0.0;              long oldTrip = 0;              long newTrip =0;              long diffMinutes = 0;              long diffHours = 0;              long diffDays = 0;              long diffSecondsnew = 0;              long diffMinutesnew = 0;              long diffHoursnew = 0;              long diffDaysnew = 0;              double oldTripDistance = 0.0;              double newTripDistance = 0.0;              double orthogonalDistance = flight.getAirportsDistance();                for (Segment seg: flight.getSegments()){                    if (seg.getStatus()==0.0){                      distanceClimb=distanceClimb+seg.getLength();                      distanceTimeClimb+=seg.getEndTime().getTime()-seg.getInitialTime().getTime();                    }                    if (seg.getStatus()==1){                      distanceDescent=distanceDescent+seg.getLength();                      distanceTimeDescent+=seg.getEndTime().getTime()-seg.getInitialTime().getTime();                    }                    if (seg.getStatus()==2){                      distanceCruise=distanceCruise+seg.getLength();                      distanceTimeCruise+=seg.getEndTime().getTime()-seg.getInitialTime().getTime();                  }                  if (found != true && seg.getStatus()==2 && seg.getEndFL()>= (flight.getMaxFL()-40)){                      found = true;                      flight.setInicioCrucero(seg.getSequence());                      System.out.println("Es el segment crucero primero!!"+ seg.getId());                    }                    if (found2 !=true && seg.getStatus()==2 && seg.getEndFL()>= (flight.getMaxFLTotal()-40) && flight.getInicioCrucero()!=seg.getSequence()){                      flight.setFinalCrucero(seg.getSequence());                      found2= true;                      System.out.println("Es el segment crucero ultimo!!"+ seg.getId());                      //break;                  }                } //end segments                System.out.println("Climb "+distanceClimb);              System.out.println("Cruise "+distanceCruise);              System.out.println("Descent "+distanceDescent);              System.out.println("distanceTimeCruise"+distanceTimeCruise);              groundSpeed = distanceCruise/distanceTimeCruise;              cruiseDistance = orthogonalDistance-distanceDescent-distanceClimb;              cruiseTime =cruiseDistance/groundSpeed;                  if (found2==false){                  flight.setFinalCrucero(flight.getInicioCrucero());                  System.out.println("Soy un único punto de crucero");              }              oldTrip = distanceTimeClimb+distanceTimeCruise+distanceTimeDescent;              newTrip= (long) (distanceTimeClimb+cruiseTime+distanceTimeDescent);                oldTripDistance = distanceCruise+distanceDescent+distanceClimb;              newTripDistance = orthogonalDistance;              flight.setOldTripDistance(oldTripDistance);              flight.setNewTripDistance(newTripDistance);                    flight.setOldTripTime(new Time (oldTrip));              flight.setNewTripTime(new Time (newTrip));              diffSeconds = oldTrip / 1000 % 60;              diffMinutes = oldTrip / (60 \* 1000) % 60;              diffHours = oldTrip / (60 \* 60 \* 1000) % 24;              diffDays = oldTrip / (24 \* 60 \* 60 \* 1000);                diffSecondsnew = newTrip / 1000 % 60;              diffMinutesnew = newTrip / (60 \* 1000) % 60;              diffHoursnew = newTrip / (60 \* 60 \* 1000) % 24;              diffDaysnew = newTrip / (24 \* 60 \* 60 \* 1000);                  //Updating status              for (Segment seg: flight.getSegments()){                  System.out.println("a"+flight.getSegments().get(0).getSequence());                  System.out.println("b"+flight.getSegments().get(1).getSequence());                  System.out.println("c"+flight.getSegments().get(2).getSequence());                  }              if (isDeparture(flight.getADEP())){                  }          }        }          public boolean isDeparture (String ADEP){          return ADEP.equals("LSZH");          }      public void insertFlight(CS3SESARImprovedFlight flight) throws SQLException {          Connection conn = SESARImprovements.createConnection();          System.out.println("I'm gonna insert");            String insertTableSQL = "INSERT INTO ground\_improvements"                  + "(FltNum, oldTripDistance, oldTripTime, improvedTripDistance,improvedTripTime,departureTime) " + "VALUES"                  + "('"+flight.getFltNum()+"','"+flight.getOldTripDistance()+"','"+flight.getOldTripTime()+"','"+flight.getAirportsDistance()+"','"+ flight.getNewTripTime()+"',"+"null"+")";            System.out.println(insertTableSQL);            conn.createStatement().executeUpdate(insertTableSQL);        }          public void insertFlightFixed(CS3SESARImprovedFlight flight, long newCruisingTime) throws SQLException {          Connection conn = SESARImprovements.createConnection();          System.out.println("I'm gonna insert");            String insertTableSQL = "INSERT INTO ground\_sesar\_improvements"                  + "(FltNum, oldTripDistance, newTripDistance,newCruisingTime,oldTripTime, newTripTime) " + "VALUES"                  + "('"+flight.getFltNum()+"','"+flight.getOldTripDistance()+"','"+flight.getAirportsDistance()+"','"+ newCruisingTime+"','"+flight.getOldTripTime()+"','"+flight.getNewTripTime()+"')";            System.out.println(insertTableSQL);            conn.createStatement().executeUpdate(insertTableSQL);        }        public void updateOriginalFlight(long cruisingTime, String fltNum) throws SQLException {          Connection conn = SESARImprovements.createConnection();          System.out.println("I'm gonna Update");            String updateCruisingTime= "Update flight set cruising\_time= '"+cruisingTime+"' where  fltNum= '"+ fltNum+"'";          System.out.println(updateCruisingTime);          conn.createStatement().executeUpdate(updateCruisingTime);        }          public void updateOriginalFlightMTT(int mtt, String fltNum) throws SQLException {          Connection conn = SESARImprovements.createConnection();          System.out.println("I'm gonna Update");            String updateCruisingTime= "Update flight set MTT= '"+mtt+"' where  fltNum= '"+ fltNum+"'";          System.out.println(updateCruisingTime);          conn.createStatement().executeUpdate(updateCruisingTime);        }        public void updateOriginalFlightFuel(CS3SESARImprovedFlight flight) throws SQLException {          Connection conn = SESARImprovements.createConnection();          System.out.println("I'm gonna Update Fuel");          String updatefuel= "Update flight set fuel\_consumption= '"+flight.getConsumption()+"' where  fltNum= '"+ flight.getFltNum()+"'";          System.out.println(updatefuel);          conn.createStatement().executeUpdate(updatefuel);        }        public void updateGroundImprovedFlightOldTrip(Long cruisingTime, String fltNum) throws SQLException {          Connection conn = SESARImprovements.createConnection();          System.out.println("I'm gonna Update Old Trip");          String updateCruisingTime= "Update flight set cruising\_time= '"+cruisingTime+"' where  fltNum= '"+ fltNum+"'";          System.out.println(updateCruisingTime);          conn.createStatement().executeUpdate(updateCruisingTime);        }          //ground + sesar        public void groundImprovementsSesar () throws SQLException {              Connection conn = SESARImprovements.createConnection();          ResultSet rsflightsDEP = null;          ResultSet pax = null;          ResultSet rsupdateDeparture = null;          ResultSet rsFlightsBefore =null;          ResultSet groundImprovements = null;          String selectpax= "SELECT connectionID ,COUNT(\*) FROM cassiopeia\_cs3.pax\_connection group by connectionID";            Date arrivalTime = new Date();          Date connectingTime = new Date ();          Date readyTime =new Date();          String matrix = " SELECT pc.connectionID, "                  + "fc.a\_FltNum ARRIVAL\_FLIGHT,"                  + " af.ARVT\_1 ESTIMATED\_ARRIVAL\_TIME,"                  + "af.ARVT\_3 ACTUAL\_ARRIVAL\_TIME, "                  + "fc.d\_FltNum DEPARTURE\_FLIGHT, "                  + "df.ARVT\_1 ESTIMATED\_ARRIVAL\_TIME, "                  + "df.ARVT\_3 ACTUAL\_ARRIVAL\_TIME, "                  + "COUNT(\*) Number\_Pax,"                  + "fc.MCT, "                  + "df.EOBT\_1 eobt, "                  + "df.CORR\_REGISTRATION REG FROM cassiopeia\_cs3.pax\_connection pc INNER JOIN cassiopeia\_cs3.flight\_connection fc ON fc.connectionID = pc.connectionID INNER JOIN cassiopeia\_cs3.flight af ON fc.a\_FltNum = af.FltNum INNER JOIN cassiopeia\_cs3.flight df ON fc.d\_FltNum = df.FltNum group by connectionID";            groundImprovements = conn.createStatement().executeQuery(matrix);          while (groundImprovements.next()){              arrivalTime = groundImprovements.getTimestamp(3);              connectingTime = DateHandler.sumTime(arrivalTime, groundImprovements.getInt(9));                System.out.println ("connecting time "+ connectingTime);              String selectFlightsBefore= "SELECT \* FROM cassiopeia\_cs3.flight where CORR\_REGISTRATION = '"+groundImprovements.getString(11) + "';" ;              rsFlightsBefore = conn.createStatement().executeQuery(selectFlightsBefore);              //System.out.println("Number of flights with the same reg "+ rsFlightsBefore.);              int i=0;              System.out.println("Estimated departure but of outbound/departure "+ groundImprovements.getTimestamp(10)+ " "+groundImprovements.getInt(1)+","+groundImprovements.getString(2)+","+groundImprovements.getTimestamp(3)+","+groundImprovements.getTimestamp(4)+","+groundImprovements.getString(5)+","+groundImprovements.getString(11));              List <Date> dates= new ArrayList <Date>();              boolean isSame=false;              while (rsFlightsBefore.next()){               if (isSame == false && rsFlightsBefore.getString("ADES").equals("LSZH")){                   if (rsFlightsBefore.getString(1).equals(groundImprovements.getString(5))){                       isSame=true;                       System.out.println("Same flight with same aircraft");                       readyTime = DateHandler.sumTime(arrivalTime, 30);                       i++;                       break;                   }                  dates.add(rsFlightsBefore.getTimestamp(10));                  i++;               }                }                System.out.println("num track of flights "+ i);              for (Date date1: dates){                  System.out.println("Before sort" +date1.toString());                }              dates = DateHandler.sortDates(dates);                for (Date date2: dates){                  System.out.println("After sort" +date2.toString());                }              Date flightBefore = new Date();              for (Date date: dates){                  System.out.println("Outbound "+ groundImprovements.getTimestamp(10));                  System.out.println("Arrival Time "+date.toString());                  if (!DateHandler.isBefore(date.toString(), groundImprovements.getTimestamp(10).toString())){                      System.out.println("Entro");                      break;                    }                  else{                      flightBefore = date;                      //Turnaroundtime fixed to 30 in Zurich WARNING ASK AB where is the data of "WK\_TBL\_CAT"!!                      readyTime = DateHandler.sumTime(flightBefore, 30);                  }                }              System.out.println ("Winner flight "+flightBefore);              System.out.println("Ready Time "+ readyTime.toString());              System.out.println("Connecting Time " + connectingTime.toString());              Date newDepartureTime = new Date();              Date newarrivalTime = new Date();              //take maximum from readyTime and ConnectingTime              if (DateHandler.isBefore(DateHandler.DatetoString(readyTime), DateHandler.DatetoString(connectingTime))){                  newDepartureTime= connectingTime;                }              else              {                  newDepartureTime=readyTime;              }                   String updateDeparture= "Update ground\_improvements set departureTime= '"+DateHandler.DatetoString(newDepartureTime)+"' where  FltNum= '"+ groundImprovements.getString(5)+"'";                 System.out.println(updateDeparture);                 conn.createStatement().executeUpdate(updateDeparture);                 newarrivalTime = DateHandler.sumTime(newDepartureTime, Utils.longToInt(DateHandler.millisecondsToMinutes((groundImprovements.getTimestamp(6).getTime()))));                 String updateArrival= "Update ground\_improvements set arrivalTime= '"+DateHandler.DatetoString(newarrivalTime)+"' where  FltNum= '"+ groundImprovements.getString(5)+"'";                 conn.createStatement().executeUpdate(updateArrival);              }        }          ...      public void trajectoryImprovementsFixed () throws SQLException{              Connection conn = SESARImprovements.createConnection();              ResultSet rssegments = null;              ResultSet rsflights = null;              ResultSet dep = null;              ResultSet arriv = null;              Date firstSegmentDate = null;              Date lastSegmentDate = null;              //Map <String, CS3SESARImprovedFlight> improvedFlights = new HashMap <String, CS3SESARImprovedFlight>();              //List <CS3SESARImprovedFlight> improvedFlights = new ArrayList <CS3SESARImprovedFlight>();              String selectFlights= "SELECT \* FROM cassiopeia\_cs3.flight";              rsflights = conn.createStatement().executeQuery(selectFlights);                while (rsflights.next()){                      String selectSegments= "SELECT \* FROM cassiopeia\_cs3.flight\_segment where FltNum = '"+ rsflights.getString(1)+"' order by sequence";                  String selectGeoLocationDepart = "SELECT latitude,longitude FROM agent WHERE idagent = '"+rsflights.getString(5)+"'";                  String selectGeoLocationArriv = "SELECT latitude,longitude FROM agent WHERE idagent = '"+rsflights.getString(6)+"'";                  CS3SESARImprovedFlight flight = new CS3SESARImprovedFlight ();                  flight.setADEP(rsflights.getString(5));                  flight.setADES(rsflights.getString(6));                    dep = conn.createStatement().executeQuery(selectGeoLocationDepart);                  arriv = conn.createStatement().executeQuery(selectGeoLocationArriv);                  double latitudeDEP = 0.0;                  double longitudeDEP = 0.0;                  double latitudeArriv = 0.0;                  double longitudeArriv = 0.0;                    if (dep.next()){                      latitudeDEP = dep.getDouble(1);                      longitudeDEP= dep.getDouble(2);                  }                  if (arriv.next()){                      latitudeArriv= arriv.getDouble(1);                      longitudeArriv = arriv.getDouble(2);                  }                      flight.setFltNum(rsflights.getString(1));                  flight.setAIRCRAFT\_TYPE\_ICAO\_ID(rsflights.getString("AIRCRAFT\_TYPE\_ICAO\_ID"));                      int maxFL=0;                  int maxFLtotal= 0;                    rssegments = conn.createStatement().executeQuery(selectSegments);                  double firstLat=0.0;                  double firstLong=0.0;                  double lastLat=0.0;                  double lastLong=0.0;                    while (rssegments.next()) {                      if (rssegments.isFirst()){                          firstSegmentDate = rssegments.getTimestamp("datetime\_begin\_segment");                          firstLat = rssegments.getDouble("lat\_begin\_segment");                          firstLong = rssegments.getDouble("lon\_begin\_segment");                        }                      if (rssegments.isLast()){                            lastSegmentDate = rssegments.getTimestamp("datetime\_end\_segment");                          lastLat = rssegments.getDouble("lat\_end\_segment");                          lastLong = rssegments.getDouble("lon\_begin\_segment");                      }                      if (maxFLtotal<=rssegments.getInt(3) ){                          maxFLtotal =  rssegments.getInt(3);                      }                        if (rssegments.getInt(5) == 2){                        }                      //get MaxFL on cruise                      if (rssegments.getInt(5) == 2 && maxFL<=rssegments.getInt(3) ){                          maxFL=rssegments.getInt(3);                      }                      Segment seg =new Segment (rssegments.getString(2), rssegments.getInt(3), rssegments.getInt(4), rssegments.getInt(10), rssegments.getInt(5), rssegments.getDouble(11), rssegments.getTime(12),rssegments.getTime(13), rssegments.getDouble(6),rssegments.getDouble(7),rssegments.getDouble(8), rssegments.getDouble(9));                      flight.addSegment(seg);                  }//end segments                    long oldTrip = lastSegmentDate.getTime()-firstSegmentDate.getTime();                  long diffSeconds = oldTrip / 1000;                  //Since it is tactic, it is not necessary                  int day = (int)TimeUnit.SECONDS.toDays(diffSeconds);                  long hours = TimeUnit.SECONDS.toHours(diffSeconds) - (day \*24);                  long minute = TimeUnit.SECONDS.toMinutes(diffSeconds) - (TimeUnit.SECONDS.toHours(diffSeconds)\* 60);                  long second = TimeUnit.SECONDS.toSeconds(diffSeconds) - (TimeUnit.SECONDS.toMinutes(diffSeconds) \*60);                  //new Time (diffSeconds)                  System.out.println( hours+":"+minute+":"+second);                    System.out.println("last segment "+ lastSegmentDate.getTime()+ "first "+ firstSegmentDate.getTime()+ "old trip "+ oldTrip);                    flight.setOldTripTime(new Time (oldTrip));                    flight.setMaxFL(maxFL);                  flight.setMaxFLTotal(maxFLtotal);                  //in miles!!!                  //haversine between first and last point                  flight.setAirportsDistance(Haversine.haversineFinal(firstLat, firstLong, lastLat, lastLong));                  this.improvedFlights.add(flight);                }                System.out.println("Number of flights "+improvedFlights.size());                for (CS3SESARImprovedFlight flightFixed: improvedFlights){                    System.out.println("Nuevo vuelo" );                  boolean found= false;                  boolean found2= false;                  double distanceClimb =0.0;                  double distanceDescent =0.0;                  double distanceCruise =0.0;                  long distanceTimeClimb = 0;                  long distanceTimeDescent = 0;                  long distanceTimeCruise = 0;                  long newCruiseTime =0;                  double groundSpeed = 0.0;                  double cruiseTime = 0.0;                  long diffSeconds = 0;                  //long diff = 0;                  double cruiseDistance = 0.0;                  double orthogonalDistance = 0.0;                    long newTrip =0;                  long diffMinutes = 0;                  long diffHours = 0;                  long diffDays = 0;                  long diffSecondsnew = 0;                  long diffMinutesnew = 0;                  long diffHoursnew = 0;                  long diffDaysnew = 0;                  long difference=0;                  double oldTripDistance = 0.0;                  double newTripDistance = 0.0;                      for (Segment seg: flightFixed.getSegments()){                      //Ascending                      if (seg.getStatus()==0.0){                          if (seg.getLength()==0.0){                              System.out.println("OJO");                          }                          distanceClimb=distanceClimb+seg.getLength();                          distanceTimeClimb+=seg.getEndTime().getTime()-seg.getInitialTime().getTime();                      }                      //Descending                      if (seg.getStatus()==1){                          if (seg.getLength()==0.0){                              System.out.println("OJO");                          }                          distanceDescent=distanceDescent+seg.getLength();                          distanceTimeDescent=+seg.getEndTime().getTime()-seg.getInitialTime().getTime();                          }                      //Calculate cruisingTime and insert to original flightPlan                      if (seg.getStatus()==2){                          if (seg.getLength()==0.0){                              System.out.println("OJO");                          }                          distanceCruise=distanceCruise+seg.getLength();                          difference =seg.getEndTime().getTime()-seg.getInitialTime().getTime();                          //difference =Math.abs(difference);                          diffSeconds = difference / 1000 % 60;                          diffMinutes = difference / (60 \* 1000) % 60;                          diffHours = difference/ (60 \* 60 \* 1000) % 24;                          diffDays = difference/ (24 \* 60 \* 60 \* 1000);                          if (seg.getInitialTime().getTime()>seg.getEndTime().getTime()){                              difference= seg.getEndTime().getTime()-seg.getInitialTime().getTime();                              System.out.println("Special case "+difference);                              Date dif = new Date();                              dif.setTime(difference);                              dif=DateHandler.sumTime(dif,  24\*60);                              diffSeconds = dif.getTime() / 1000 % 60;                              diffMinutes = dif.getTime()  / (60 \* 1000) % 60;                              diffHours = dif.getTime() / (60 \* 60 \* 1000) % 24;                              diffDays = dif.getTime() / (24 \* 60 \* 60 \* 1000);                              System.out.println("Special case after "+ " "+diffDays+" "+ diffHours+ " "+diffMinutes+" "+ diffSeconds+"original "+ dif);                              distanceTimeCruise+=dif.getTime();                              System.out.println("Special change "+distanceTimeCruise+" "+diffDays+" "+ diffHours+ " "+diffMinutes+" "+ diffSeconds);                            }                          else {                              distanceTimeCruise+=seg.getEndTime().getTime()-seg.getInitialTime().getTime();                            diffSeconds = distanceTimeCruise / 1000 % 60;                          diffMinutes = distanceTimeCruise / (60 \* 1000) % 60;                          diffHours = distanceTimeCruise / (60 \* 60 \* 1000) % 24;                          diffDays = distanceTimeCruise / (24 \* 60 \* 60 \* 1000);                          System.out.println("Change"+distanceTimeCruise+" "+diffDays+" "+ diffHours+ " "+diffMinutes+" "+ diffSeconds);                          }                          }                      if (found != true && seg.getStatus()==2 && seg.getEndFL()>= (flightFixed.getMaxFL()-40)){                          found = true;                          flightFixed.setInicioCrucero(seg.getSequence());                          System.out.println("Es el segment crucero primero!!"+ seg.getId());                        }                        //System.out.println("Found + Found 2"+ found+ " "+ found2);                      if (found2 !=true && seg.getStatus()==2 && seg.getEndFL()>= (flightFixed.getMaxFLTotal()-40) && flightFixed.getInicioCrucero()!=seg.getSequence()){                          flightFixed.setFinalCrucero(seg.getSequence());                          found2= true;                          System.out.println("Es el segment crucero ultimo!!"+ seg.getId());                          //break;                        }                        oldTripDistance+=Haversine.haversineFinal(seg.getInitialLatitude(), seg.getInitialLongitude(), seg.getFinalLatitude(), seg.getFinalLongitude());                  } //end segments                  diffSeconds = distanceTimeCruise / 1000 % 60;                  diffMinutes = distanceTimeCruise / (60 \* 1000) % 60;                  diffHours = distanceTimeCruise / (60 \* 60 \* 1000) % 24;                  diffDays = distanceTimeCruise / (24 \* 60 \* 60 \* 1000);                  System.out.println("cruising time long "+ distanceTimeCruise+" cruising time testing "+ diffSeconds+"sec "+diffMinutes+  "min "+ diffHours+ "hours"+  "for flight "+ flightFixed.getFltNum());                      if (found2==false){                      flightFixed.setFinalCrucero(flightFixed.getInicioCrucero());                      System.out.println("Soy un único punto de crucero");                  }                    newTrip= (long) (distanceTimeClimb+cruiseTime+distanceTimeDescent);                      flightFixed.setOldTripDistance(oldTripDistance);                  flightFixed.setNewTripDistance(flightFixed.getAirportsDistance());                  flightFixed.setNewTripTime(new Time (newTrip));                      diffSecondsnew = newTrip / 1000 % 60;                  diffMinutesnew = newTrip / (60 \* 1000) % 60;                  diffHoursnew = newTrip / (60 \* 60 \* 1000) % 24;                  diffDaysnew = newTrip / (24 \* 60 \* 60 \* 1000);                    this.insertFlight(flightFixed);                  newCruiseTime = flightFixed.getOldTripTime().getTime()-distanceTimeClimb-distanceTimeDescent;                  if (!Utils.isZero(distanceTimeCruise)){                      //this.updateOriginalFlight(distanceTimeCruise,flightFixed.getFltNum());                  }                    //this.insertFlightFixed(flightFixed, newCruiseTime);                  distanceTimeCruise= 0;              }//end for flightFixed          }        public List <CS3SESARImprovedFlight> fuelModel() throws SQLException{          List <CS3SESARImprovedAircraft> aircrafts = ExcelHandler.getModelFromExcel();              List <CS3SESARImprovedAircraft>  newAircrafts= new ArrayList <CS3SESARImprovedAircraft>();            for (CS3SESARImprovedAircraft newAircraft: aircrafts){              if (!Utils.isEmpty(newAircraft.getType()) && !Utils.isZeroDouble(newAircraft.getP1())){                  newAircrafts.add(newAircraft);              }            }            for (CS3SESARImprovedFlight flight: this.improvedFlights){                double consumption= 0.0;                for (CS3SESARImprovedAircraft aircraft: newAircrafts){                if (flight.getAIRCRAFT\_TYPE\_ICAO\_ID().equals(aircraft.getType()) && Utils.isBetween(aircraft.getDistMin(), aircraft.getDistMax(),flight.getOldTripDistance())){                    consumption=aircraft.getP1()\* Math.pow(aircraft.getSmin(),4)+ aircraft.getP2()\* Math.pow(aircraft.getSmin(), 3)+aircraft.getP3()\*Math.pow( aircraft.getSmin(), 2)+ aircraft.getP4()\*aircraft.getSmin()+aircraft.getP5()  ;                  flight.setConsumption(consumption);                  this.updateOriginalFlightFuel(flight);              }            }            }            return this.improvedFlights;      }        public List <CS3SESARImprovedFlight> fuelModelII() throws SQLException{          List <CS3SESARImprovedAircraft> aircrafts = ExcelHandler.getModelFromExcelII();          for (CS3SESARImprovedFlight flight: this.improvedFlights){                for (CS3SESARImprovedAircraft aircraft: aircrafts){                if (flight.getAIRCRAFT\_TYPE\_ICAO\_ID().equals(aircraft.getType())){                  flight.setConsumption(aircraft.getConsumption());                  this.updateOriginalFlightFuel(flight);              }            }            }          //note: 180 flights added for fuel models from AB (INNAXIS :) )          return this.improvedFlights;      }          public int calculateMTT(ResultSet flight) throws SQLException {          // TODO Extend turnaround calculation (40 min)          int time = 2400000;          //String model = flight.getString(3);          String model = flight.getString(3);          //Airport dep\_ap  = current.getDep\_airport();          //Airline carrier = current.getCarrier();          String depAirp = flight.getString(5);          String arrivAirp = flight.getString(6);          String ao = flight.getString(2);          String aicraftType = flight.getString(3);          //AgentType ao\_type = AgentType.subtype(carrier.getSubtype());          //switch(ao\_type) {          ResultSet agentid = null;          ResultSet agentname = null;          ResultSet tblcategory = null;          String selectclass =  "SELECT idclass FROM agent WHERE idagent = '"+ao+"'";          agentid = conn.createStatement().executeQuery(selectclass);          String selectType =   "SELECT classname FROM agent WHERE idclass = '"+agentid.getInt(1)+"'";          String selectaircraft=   "SELECT WK\_TBL\_CAT FROM cassiopeia\_cs3.aircraft where MODEL = '"+aicraftType+"'";          agentname =  conn.createStatement().executeQuery(selectType);          tblcategory = conn.createStatement().executeQuery(selectaircraft);          switch(agentname.getString(1)) {          case "FSC":                //note: I think this is to ensure airport is ZRH. IL: AB said yes.                if (depAirp.equals("LSZH")){                  if (ao.equals("SWR")){                      if (tblcategory.getString(1).equals('H')) {                          time = 90 \* 60000;                      }                      else if (tblcategory.getString(1).equals('M')) {                          time = 23 \* 60000;                      }                  }                  else{                      if (tblcategory.getString(1).equals ('J')) {                          time = 180 \* 60000;                      }                      else if (tblcategory.getString(1).equals ('H')) {                          time = 102 \* 60000;                      }                      else if (tblcategory.getString(1).equals ('M')) {                          time = 22 \* 60000;                      }                  }              }              else {                  if (tblcategory.getString(1).equals('H')) {                      time = 96 \* 60000;                  }                  else if (tblcategory.getString(1).equals('M')) {                      time = 22 \* 60000;                  }              }              break;                case "LCC":              if (depAirp.equals("LSZH")) {                  if (tblcategory.getString(1).equals('H')) {                      time = 80 \* 60000;                  }                  else if (tblcategory.getString(1).equals('M')) {                      time = 26 \* 60000;                  }              }              else {                  if (tblcategory.getString(1).equals('H')) {                      time = 80 \* 60000;                  }                  else if (tblcategory.getString(1).equals('M')) {                      time = 22 \* 60000;                  }              }              break;          case "REG":              if (depAirp.equals("LSZH")) {                  time = 17 \* 60000;              }              else {                  if (tblcategory.getString(1).equals('H')) {                      time = 80 \* 60000;                  }                  else if (tblcategory.getString(1).equals('M')){                      time = 18 \* 60000;                  }              }              break;          case "CHT":              if (depAirp.equals("LSZH")) {                  time = 37 \* 60000;              }              else {                  if (tblcategory.getString(1).equals('H')) {                      time = 80 \* 60000;                  }                  else if (tblcategory.getString(1).equals('M')) {                      time = 40 \* 60000;                  }              }              break;          default:              break;          }          return time;      }          //vers 2.0      public int calculateMTT2 () throws SQLException {          // TODO Extend turnaround calculation (40 min)          Connection conn = SESARImprovements.createConnection();          ResultSet rsflights = null;          String selectFlights= "SELECT \* FROM cassiopeia\_cs3.flight";          rsflights = conn.createStatement().executeQuery(selectFlights);            int time = 2400000;          while (rsflights.next()){          String depAirp = rsflights.getString(5);          String ao = rsflights.getString(2);          String aicraftType = rsflights.getString(3);          ResultSet agentid = null;          ResultSet agentname = null;          ResultSet tblcategory = null;          String selectType = null;          String selectclass =  "SELECT idclass FROM agent WHERE idagent = '"+ao+"'";          agentid = conn.createStatement().executeQuery(selectclass);          if (agentid.next()){              selectType =   "SELECT classname FROM agent\_class WHERE idclass = '"+agentid.getInt(1)+"'";              agentname =  conn.createStatement().executeQuery(selectType);          }          String selectaircraft=   "SELECT WK\_TBL\_CAT FROM cassiopeia\_cs3.aircraft where MODEL = '"+aicraftType+"'";          tblcategory = conn.createStatement().executeQuery(selectaircraft);          if (agentname.next() && tblcategory.next()){          switch(agentname.getString(1)) {          case "FSC":                  if (depAirp.equals("LSZH")){                  if (ao.equals("SWR")){                      if (tblcategory.getString(1).equals('H')) {                          time = 90 \* 60000;                      }                      else if (tblcategory.getString(1).equals('M')) {                          time = 23 \* 60000;                      }                  }                  else{                      if (tblcategory.getString(1).equals ('J')) {                          time = 180 \* 60000;                      }                      else if (tblcategory.getString(1).equals ('H')) {                          time = 102 \* 60000;                      }                      else if (tblcategory.getString(1).equals ('M')) {                          time = 22 \* 60000;                      }                  }              }              else {                  if (tblcategory.getString(1).equals('H')) {                      time = 96 \* 60000;                  }                  else if (tblcategory.getString(1).equals('M')) {                      time = 22 \* 60000;                  }              }              break;                case "LCC":              if (depAirp.equals("LSZH")) {                  if (tblcategory.getString(1).equals('H')) {                      time = 80 \* 60000;                  }                  else if (tblcategory.getString(1).equals('M')) {                      time = 26 \* 60000;                  }              }              else {                  if (tblcategory.getString(1).equals('H')) {                      time = 80 \* 60000;                  }                  else if (tblcategory.getString(1).equals('M')) {                      time = 22 \* 60000;                  }              }              break;          case "REG":              if (depAirp.equals("LSZH")) {                  time = 17 \* 60000;              }              else {                  if (tblcategory.getString(1).equals('H')) {                      time = 80 \* 60000;                  }                  else if (tblcategory.getString(1).equals('M')){                      time = 18 \* 60000;                  }              }              break;          case "CHT":              if (depAirp.equals("LSZH")) {                  time = 37 \* 60000;              }              else {                  if (tblcategory.getString(1).equals('H')) {                      time = 80 \* 60000;                  }                  else if (tblcategory.getString(1).equals('M')) {                      time = 40 \* 60000;                  }              }              break;          default:              break;          }          }          this.updateOriginalFlightMTT(time, rsflights.getString(1));          }          return time;      }      public double haversineBetween2Points(double latitude1, double longitude1, double latitude2, double longitude2){            return Haversine.distFrom(latitude1,longitude2, latitude2, longitude2);      }  } |

# References

Apache commons, <http://commons.apache.org/proper/commons-math/>, last visited May 2015.

A. Collinot, A. Drogoul, and P. Benhamou, “Agent oriented design of a soccer robot team,” in Proceedings of the 2nd International Conference on Multi-Agent Systems (ICMAS-96), Kyoto, Japan, 1996, pp. 41–47

Caire, G., Coulier, W., Garijo, F., Gomez, J., Pavon, J., Leal, F., Chainho, P., Kearney, P., Stark, J., Evans, R., and Massonet, P. (2002), “Agent-oriented analysis using MESSAGE/UML”, Proceedings of the 2nd International Workshop on Agent-Oriented Software Engineering. Lecture Notes in Computer Science, Vol. 2222. pp. 119–135, Springer Verlag.

F. Giunchiglia, J. Mylopoulos, and A. Perini. The tropos software development methodology: Processes, models and diagrams. In Third International Workshop on Agent-Oriented Software Engineering, July 2002.

F. Mark, A and Wood Scott. DeLoach: an overview of the multi-agent systems engineering methodology. In: Ciancarini, P., Wooldridge, M. (Eds.), Proceedings of the First International Workshop on Agent-Oriented Software Engineering, Lecture Notes in Computer Science, vol. 1957. Springer, Berlin.

FIPA Specs, http://www.fipa.org/specs/fipa00061/index.html, last visited May 2015.

G. Caire, W. Coulier, F. Garijo, J. Gomez, J. Pavon, F. Leal, P. Chainho, P. Kearney, J. Stark, R. Evans, and P. Massonet, “Agent Oriented Analysis Using Message/UML,” Lecture Notes in Computer Science, M. Wooldridge, G. Weiss, and P. Ciancarini (Eds.), Springer-Verlag, vol. 2222, 2002, pp. 119-135.

L. Padgham, M. Winikoff.: Prometheus: A methodology for developing intelligent agents. In: Third International Workshop on Agent-Oriented Software Engineering, July 2002.

M. Bratman, Intentions, Plans, and Practical Reason (Harvard University Press, Cambridge, MA, 1987).

M. Wooldridge, N. R. Jennings, and D. Kinny, “The gaia methodology for agent-oriented analysis and design,” Autonomous Agents and Multi-Agent Systems, vol.3(3), pp. 285-312, 2000.

 M. Wooldridge, “Intelligent agents,” in Multi-Agent Systems, M. Wooldridge and G. Weiss, Eds., pp. 3–51, MIT Press, Cambridge, Mass, USA, 1999

F. Bellifemine, G. Rimassa, and A. Poggi. JADE – A FIPA-compliant agent framework. In 4th International Conference on the Practical Applications of Agents and Multi-Agent Systems (PAAM-99), pages 97–108, London, UK, December 1999.

M. Bratman. Intention, Plans, and Practical Reason. Harvard University Press, Cambridge, Massachusetts, 1987.

Pokahr, A., Braubach, L., Lamersdorf, W. (2003) Jadex: Implementing a BDI-Infrastructure for JADE Agents. EXP – in search of innovation 3: pp. 76-85