Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FIN DE GRADO

TÍTULO DEL TFG: Entorno de simulación basado en agentes para A-CDM (Airport Collaborative Decision Making)

TITULACIÓN: Grau en Ingeniería de Aeronavegación

AUTORES: Ivan Garcia Vasco
Xavier Morell Llorens

DIRECTOR: Miguel Valero Garcia
Xavier Prats Menéndez
Luís Delgado Muñoz

FECHA: 6 de Febrero de 2015

Título: Entorno de simulación basado en agentes para A-CDM (Airport Collaborative Decision Making)

Autores: Ivan Garcia Vasco
Xavier Morell Llorens

Director: Miguel Valero Garcia
Xavier Prats Menéndez
Luís Delgado Muñoz

Fecha: 6 de Febrero de 2015

Resumen

Debido al constante aumento del tráfico aéreo en Europa, los aeropuertos se ven obligados a trabajar con un volumen de vuelos próximo a la máxima capacidad que pueden operar. Por este motivo es necesario tomar medidas para mejorar la eficiencia y la capacidad de los aeropuertos. Para mejorar la su capacidad, los diferentes actores que intervienen en el aeropuerto deben comunicarse entre ellos con el objetivo de mejorar la manera de administrar el tráfico.
Airport Collaborative Decision Making (A-CDM) se basa en compartir la información. Un mayor uso de los recursos es posible cuando los diferentes stakeholders del aeropuerto comparten de manera precisa y constante la información. Una de las mayores dificultades relacionadas con la idea de compartir información entre los diferentes stakeholders es la gran cantidad de agentes que actúan en un aeropuerto y los intereses que tienen cada uno de ellos: Operadores Aéreos, compañías de ground handling, autoridades del aeropuerto, air traffic control y el Central Flow Management Unit. Es de suma importancia cuantificar los beneficios que conlleva el uso de los procedimientos A-CDM, con el objetivo de que todas convencer a estas organizaciones para que participen en el proyecto. Para conseguirlo, las simulaciones son necesarias para analizar de manera global todo el sistema y poder dar resultados.

Este proyecto presenta el desarrollo de un entorno de simulación basado en agentes, que permite realizar un análisis de los comportamientos de los diferentes actores del aeropuerto. El simulador representa explícitamente los diferentes stakeholders involucrados en un A-CDM y las interacciones entre ellos a lo largo de los 16 milestones definidos por EUROCONTROL en su manual de implementación de A-CDM, también se han implementado diferentes funcionalidades para dar realismo a las simulaciones, por ejemplo es posible añadir retrasos y condiciones adversas al tráfico aéreo para estudiar los resultados de diferentes simulaciones. Este entorno de simulación permite un desarrollo gradual e independiente de los comportamientos y de optimización de los agentes, y permite un gradual aumento de la complexidad y la fidelidad de las simulaciones.

Title: Agent-based simulation framework for A-CDM (Airport Collaborative Decision Making)

Author: Ivan Garcia Vasco
Xavier Morell Llorens

Director: Miguel Valero Garcia
Xavier Prats Menéndez
Luís Delgado Muñoz

Date: February 6, 2015

Overview

Due to the increased air traffic in Europe, airports have to handle traffic volumes near to its maximum capacity. This makes it mandatory to take measures to improve the efficiency and capacity of airports. To improve the capacity of airports, the different actors in them must communicate with each other in order to improve the way they manage the traffic.

Airport Collaborative Decision Making (A-CDM) is based on information sharing. A better use of resources can be attained when the different stakeholders at airport operations share their more accurate and updated information. One of the main difficulties when dealing with this information sharing concept is the number of stakeholders involved and their different interest and behaviour: aircraft operators, ground handling companies, airport authority, air traffic control and the Central Flow Management Unit. It is paramount to quantify the benefit of an airport collaborative decision making strategy in order to involve all these different organisations. Simulations are required to analyse the overall system and its emerging behaviour.

This project presents the development and initial testing of an agent-based framework, which allows this behavioural analysis to be done. The simulator explicitly represents the different stakeholders involved in the A-CDM and the interactions between them during the 16 milestones defined by EUROCONTROL on its A-CDM implementation manual, also different functionalities have been implemented to add realism to the simulations, such as the possibility to add delays and adverse conditions to the air traffic and to analyse the different results of the different simulations.

This framework allows independent gradual development of local behaviours and optimisation, and a gradual increase on complexity and fidelity on the simulations.

ÍNDICE GENERAL

ÍNDICE APÉNDICES

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

En esta memoria se pretende plasmar el proceso realizado para implementar un entorno de simulación basado en agentes para simular los procesos y la evolución del tráfico aéreo en un aeropuerto con un A-CDM (Airport Collaborative Decision Making) implementado.

Primero se expondrá el sistema que se quiere simular y las metodologías que se han utilizado para definir la implementación de este. Seguidamente explicaremos el proceso que se ha seguido para desarrollar este software multiagente, desde la extracción de los requisitos, pasando por el proceso de diseño, explicando las herramientas que se han utilizado.

Por otra parte se explicará en profundidad las características del programa, así como su estructura, agentes, como se ejecuta y los datos que utiliza y genera. Además, también se ha explicado la evolución que ha seguido el programa y se ha añadido una guía de cómo ejecutar y utilizar el programa.

Debido a que este trabajo ha sido pensado como un proyecto que se continuará por otros alumnos, se han añadido muchas referencias y explicaciones del código, para así facilitar la comprensión al siguiente programador.

Por último, y como ejemplo de cómo utilizar el programa y las posibilidades que este da al usuario, se ha añadido un análisis de resultados, así como unas pequeñas conclusiones.

GLOSARIO ABREVIATURAS

ABM	Agent Based Modeling
ACDM	Airport Collavorative Decision Making
A-CDM	Airport Collavorative Decision Making
ACGT	Actual Commence Ground handling Time
ACL	Agent Comunicaction Language
ADEP	Aeropuerto Departure
ADES	Aeropuerto Destination
AID	Agent Identification
AO	Aircraft Operator
ARDT	Aircraft Ready Time
ASBT	Actual Start Boarding time
ASRT	Actual Start-up Request Time
ATC	Air Traffic Controller
ATFM	Air Traffic Flow Management
ATM	Air Traffic Management
CDM	Collaborative Decision Making
CFMU	Central Flow Management Unit
CPU	Central Processing Unit
CTOT	Control Take Off Time
DDR2	Data Demand Repository
EET	Estimated Elapsed Time
EIBT	Estimated In Block Time
ELDT	Estimated Landing Time
EOBT	Estimated Off Block Time
ETOT	Estimated Take Off Time
ETOT’	Estimated Take Off Time (origin airport)
FIPA	Foundation for Intelligent Physical Agents
FIR	Flight Information Region
FL	Flight Level
FP	Flight Plan
FUM	Flight Update Message
GH	Ground Handling
H	Heavy
ICAO	International Civil Aviation Organization
ICRAT	International Conference on Research in Air Transportation
ID	Identification
IS	Information Sharing
JADE	Java Agent Development Enviroment
L	Light
MAS	Multi Agent System
M	Medium
MT	Milestone Trigger

MTTT	Minimum Turn-round Time
OP/H-IN	Operaciones hora de entrada
OP/H-OUT	Operaciones hora de entrada
SID	Standard Instrument Departure
TAPP	Tiempo de inicio deAproximación Final
TFIR	Tiempo de entrada en FIR
TOBT	Target Off Block Time
TSAT	Target Start-up Approval Time
TTOT	Target Take Off Time
V&V	Validación y Verificación
WP	Waypoint

CAPITULO 1. MOTIVACIÓN DEL PROYECTO

El objetivo principal de este proyecto es la implementación de un simulador de los procesos de un Airport Collaborative Decision Making, para poder analizar el efecto que tiene sobre el tráfico aéreo el uso de esta herramienta en un aeropuerto. Por otra parte, debido a la naturaleza de este sistema, se ha realizado toda esta implementación a través de un sistema basado en agentes.

Debido a la gran inversión, de tiempo y dinero, que conlleva la implementación de un A-CDM en un aeropuerto, se ha buscado realizar este simulador para ayudar a los responsables de los aeropuertos a hacer simulaciones y ver los resultados que se obtendrían aplicando dichas políticas y estrategias, de esta manera se puedan valorar los pros y los contras de esta inversión antes de iniciarla.

A causa de la complejidad del sistema, y de la naturaleza del mismo, se ha decidido por utilizar ABM (Modelado Basado en Agentes), que nos permite dar más realismo a las simulaciones, así como capturar las comunicaciones que se llevan a cabo, parte importante a la hora de analizar la implementación en aeropuerto real.

El hecho de que un A-CDM sea un sistema tan grande y complejo, que incluye a tantos actores, cada uno con una manera de actuar diferente, hace que en este proyecto se haya considerado asentar las bases de lo que deberá ser un simulador más grande y completo en un futuro. El trabajar con ABM también permite que haya un desarrollo gradual e independiente de cada uno de los agentes por separado.

De esta manera, se ha trabajado para asentar las bases de este entorno de simulación basado en agentes: los diferentes tipos de agentes, las relaciones entre ellos, las comunicaciones y la manera de tratar los datos.

Por otra parte, además de trabajar en las bases del entorno de simulación, se ha trabajado en dotar de realismo a las simulaciones analizando los comportamientos de los diferentes actores que influyen en el sistema CDM, así como sus responsabilidades y como repercuten sus acciones en el tráfico aéreo del aeropuerto.

También se ha hecho un trabajo extra preparando la posibilidad de añadir diferentes situaciones o “retrasos” a la simulación, así como aleatoriedad a las simulaciones.

Finalmente, gracias a la forma de simular el A-CDM, los resultados que brinda el entorno de simulación permiten al usuario hacer un análisis extenso del estado del tráfico aéreo en cualquier momento de la simulación. De esta manera se ha preparado un pequeño ejemplo de cómo se podrían analizar estos resultados, dando la posibilidad de ver cómo evolucionan los retrasos de los aviones durante toda una simulación.

CAPITULO 2. ¿Qué deseamos implementar? AIRPORT COLLABORAIVE DECISION MAKING

2.1 *Introducción, situación actual*

En la actualidad, la mayoría de los vuelos, sobre todo los vuelos cortos, tardan más que lo que se tardaba hace 30 años. ¿Cómo es posible? Teniendo en cuenta que los aviones son más rápidos que nunca, podríamos decir que el problema está en otro sitio.

De hecho, el tráfico aéreo está creciendo tan rápidamente que los recursos del aeropuerto no pueden seguir el ritmo de la demanda. En el gráfico que encontramos en la figura 2-1 podemos ver como los aeropuertos se están convirtiendo en el cuello de botella de la red de transporte aéreo [1]. Son datos un poco antiguos, pero nos sirven para ver la tendencia de la distribución del delay actualmente.

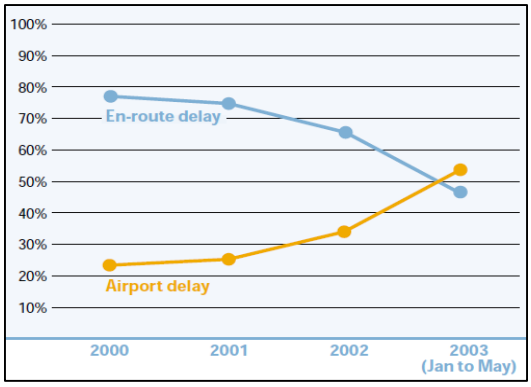


Figura 2-1: Distribución delay por año

Con el aumento del tráfico, la carga de trabajo de operadores y planificadores está aumentando, haciendo su tarea de garantizar la seguridad y la eficiencia de las operaciones más difícil cada año. En consecuencia, los vuelos toman más tiempo y los retrasos son frecuentes.

Mirando más de cerca, podemos ver que muchos problemas están relacionados con la ineficiencia en las operaciones aeroportuarias diarias y la falta de disponibilidad de información fiable. Esto hace que los diferentes actores que están involucrados en estas operaciones, estén preocupados por diferentes aspectos [1]:

- ATC
  - Congestión en aproximación y taxiway.
  - Sobrecarga de tráfico y carga de trabajo.
  - Retraso en la información dificulta la flexibilidad de planificado
- Aerolíneas
  - Poca puntualidad debido a los retrasos imprevistos.
  - Efecto en cadena de los retrasos del tráfico aéreo.
  - Utilización ineficiente de la flota.
  - Perdida de conexiones de vuelos.
  - Preferencias y prioridades no consideradas.
- Empresas de Handling

  - Poca previsibilidad debida a retrasos imprevistos
  - Uso ineficiente de sus recursos
- Aeropuerto
  - Uso ineficiente de las infraestructuras aeroportuarias limita el rendimiento del aeropuerto
- CFMU
  - Difícil cumplimiento de los slots CTOT definidos.
  - Predicciones de carga de tráfico inexactas, que resulta en un exceso o una infrautilización de la capacidad.

De hecho, lo que todas estas preocupaciones tienen en común se puede describir como:

Todos los actores carecen de información de la situación global del aeropuerto debido al flujo inadecuado e incompleto de la información.

Las razones de que esta sea la situación del aeropuerto podrían ser:

- La información más importante existe en algún lugar o sistema, pero no está realmente disponible para todos los actores.
- La información que tienen diferentes sistemas se ha desarrollado y calculado independientemente y, por tanto, puede no ser la misma.
- Algunos actores son reacios a compartir cierta información "comercialmente sensible", lo que hace que haya una restricción en la información compartida.

Por todas estas razones, desde Eurocontrol se ha llegado a la conclusión de que es necesario implementar una serie de políticas de actuación, a las que se ha llamado Airport Collaborative Decision Making.

2.2 *Airport Collaborative Decision Making*

Airport Collaborative Decision Making consiste en utilizar una serie de procedimientos y sistemas para asistir a tomar mejores decisiones basadas en el conocimiento de toda la información de las operaciones del aeropuerto. Tiene como objetivo mejorar la gestión del flujo del tráfico aéreo en los aeropuertos mediante la reducción de los retrasos, la mejora de la previsibilidad de los acontecimientos y la optimización de la utilización de los recursos [2].

De una manera muy simplificada, se puede explicar el ACDM como una mejora en la manera de administrado de la información, haciendo que todos los actores compartan toda la información que tienen sobre el tráfico aéreo en el aeropuerto. Esta información es compartida por todos los actores implicados: aerolíneas, empresas de handling, autoridades aeroportuarias, controladores (ATC), y autoridades que administran el espacio aéreo (CFMU); estos son los llamados *stakeholders* del aeropuerto y son lo que se han definido como los actores que participan en este ACDM [3].

Aeropuerto Collaborative Decision Making está ahora integrado en el concepto operacional ATM como un habilitador importante que mejorará la eficiencia operativa, la previsibilidad y puntualidad en el tráfico aéreo y los stakeholders del aeropuerto. Se espera que el ACDM tendrá un impacto en la eficiencia operativa

de los socios del aeropuerto, y puede llegar a contribuir a mejorar la planificación de recursos y tiempos de vuelo debido a una mayor previsibilidad. La aplicación de ACDM va a transformar muchas de las políticas de comunicación y los procedimientos que han dominado históricamente las operaciones aeroportuarias, trayendo mejoras sustanciales a todos los socios [4].

Como su nombre indica, ACDM se trata de actores que trabajan juntos y toman decisiones sobre la base de información precisa y de mayor calidad, donde cada bit de información tiene el mismo significado exacto para todos los socios involucrados. Un uso más eficiente de los recursos y la mejora de la puntualidad de los eventos, así como la previsibilidad son los objetivos.

En ausencia de ACDM, las decisiones operativas a menudo pueden ser incorrectas, o no se envían a todos. Los socios pueden tomar decisiones en conflicto como resultado de la falta de información o de la recepción de la información que tiene sentido divergente a diferentes socios. Abordar estas deficiencias individualmente traerá mejoras pero podemos hablar de Aeropuerto CDM. Una vez este ACDM está definido, se debería de conseguir:

- Mejora sustancial de la calidad de los datos.
- Información compartida instantáneamente entre todos los *stakeholders*.
- Entendimiento de la información por parte de todos, siguiendo una estructura para que toda la información se siga un formato conocido.
- Funciones comunes que permitan la toma de decisiones, que se definan y se implementan utilizando la información compartida para apoyar las decisiones ATM.

Estos beneficios se pueden describir como beneficios más generales, más enfocados a los datos que genera el aeropuerto. Los beneficios en temas de ATM, directamente en las operaciones del aeropuerto, los podemos encontrar resumidos en la figura 2-2 [4].

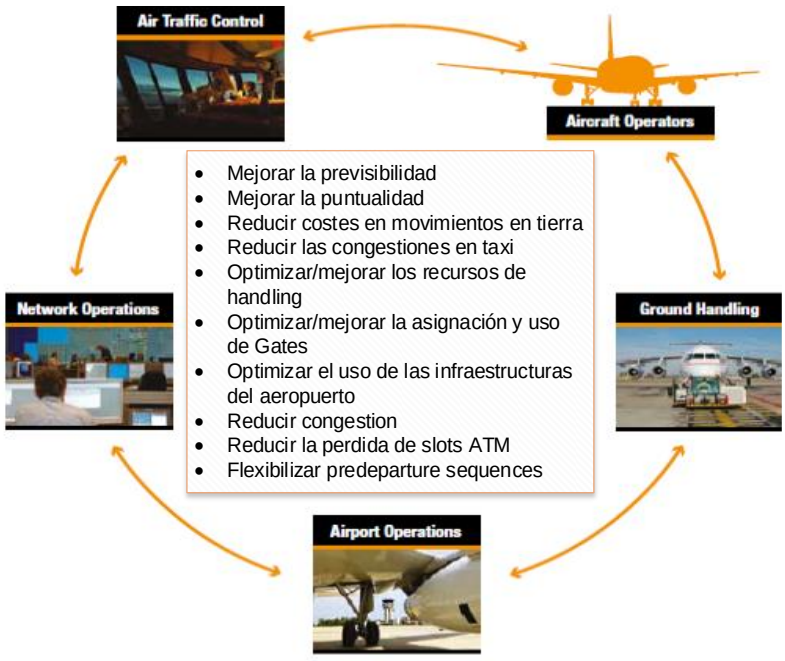


Figura 2-2: Stakeholders y objetivos ACDM

A-CDM se ha introducido en Europa durante los ensayos de campo en algunos aeropuertos y se estima que podría proporcionar rápidamente unos beneficios donde todos los afectados ganen, con beneficios sustanciales alcanzados en un período relativamente bajo retorno de la inversión [5].

Actualmente Eurocontrol proporciona una lista de los aeropuertos con un A-CDM totalmente implementado y operativo, a la que se acaba de añadir el aeropuerto de Venecia [6]. La lista es:
Berlín Schönefeld, Bruselas, Düsseldorf, Frankfurt, Helsinki, London Gatwick, London Heathrow, Madrid, Milán Malpensa, Múnich, Paris CDG, Oslo, Rome Fiumicino, Stuttgart, Zurich y Venecia.

Sin embargo, el efecto real sobre las operaciones diarias de la aplicación A-CDM para un aeropuerto determinado es difícil de cuantificar. El número de actores involucrados y la cantidad de información que se intercambia es alta y compleja. Por lo tanto, es muy difícil predecir el impacto en el sistema que algunas de estas decisiones podría producir en un entorno tan complejo y dinámico.

Impacto sobre los *stakeholders*
Cuando ACDM se presenta como proyecto en un aeropuerto, los actores tienen que reunirse y discutir el impacto y la organización de un proyecto de este tipo. Además, tienen que preparar sus propias organizaciones, y cómo se organizarán la cooperación entre ellos.
Esto es debido a que la implementación de este nuevo sistema conlleva, para los *stakeholders* del aeropuerto, dos grandes cambios:

- Nuevos procedimientos, un aumento del trabajo al tener que registrar la información y compartirla.
- Cambio de la mentalidad en las empresas y trabajadores.

La implementación de un ACDM incluye todo un conjunto de nuevos procedimientos y procesos, y una fase de entrenamiento para todo el personal. Este enfoque conjunto en nuevos procedimientos de trabajo a su vez permitirá múltiples perspectivas de las actividades de las personas y de las organizaciones individuales, y evaluar tanto el impacto individual y colectivo de los nuevos procedimientos de trabajo. Esto suele suponer un gasto adicional para las empresas, por eso es necesario un cambio de mentalidad y que todos los actores entiendan las ventajas de este sistema, y estén de acuerdo en trabajar de esta manera.

Cuando se trata de la integración de la tecnología existente, o el desarrollo de nuevas aplicaciones de automatización, se necesitan ingenieros en las discusiones para entender los problemas de funcionamiento y para poder extrapolar el impacto técnico sobre los individuos y las organizaciones.

Todos estos procesos de entrenamiento de trabajadores, nuevos procedimientos de trabajo y análisis de beneficios para empresas se pueden encontrar en el manual de implementación de ACDM redactado por Eurocontrol [4]; en esta memoria nos hemos centrado en una implementación más técnica, y no tendremos en cuenta estos apartados.

2.3 *Implementación A-CDM*

ACDM se implementa en el entorno del aeropuerto a través de la introducción de una serie de procesos que se describen en términos de:

- Reglas y procedimientos
- Requisitos (estructura) de datos de entrada
- Requisitos (estructura) de datos de salida
- Requisitos de las interfaces hombre-maquina

Reglas y procedimientos describen lo que se debe hacer con la información recibida, que información de salida hay que generar y enviar, y qué acciones emprender en respuesta a la información o eventos específicos. Los requisitos de entrada y salida describen como tiene que ser la información de los procesos.

Para esta implementación Eurocontrol define una serie de pasos a seguir, para llegar a un ACDM totalmente implementado y operativo.

- **Information Sharing**: Una plataforma que administra y centraliza la información del CDM. Administra los datos en una estructura predefinida y mantiene la información disponible para que todos los actores puedan acceder a ella.
- **Milestone Approach**: Se divide todo el proceso que realiza un avión (desde que inicia el vuelo que llega al aeropuerto hasta que despega de este) en una serie de eventos, llamados *Milestones*; para esquematizar este proceso y poder llevar un control más fácilmente.
- **Cálculo taxi time variable**: Implementación que ayuda a mejorar la predictibilidad de los tiempos, calculando el tiempo que tardará un avión (en ir de pista a gate y viceversa) dependiendo de múltiples factores.
- **Pre-departure sequence**: Procedimientos que lleva a cabo ATC para, gracias a conocer de antemano los tiempos de despegue de cada avión, ordenar las salidas. Así evitar congestiones en taxi y colas de aviones en cabecera de pista esperando a despegar.
- **Collaborative Management of FUM**: Utilización de los mensajes FUM (Flight Update Message) para el intercambio de información.
- **CDM en condiciones adversas**: Preparar los procedimientos del ACDM para administrar correctamente situaciones fuera del normal desarrollo de las operaciones aéreas, por ejemplo casos de accidente aéreo, gran nevada, problemas graves en las pistas, etc.

De todos estos pasos de la implementación total del ACDM, en el simulador nos hemos centrado en implementar la plataforma de control de la información, el Information Sharing; el control de los aviones a partir de conocer el momento del vuelo en el que se encuentran, Milestone approach; y también se ha programado la inteligencia para que ATC sea capaz de realizar correctamente una distribución de los aterrizajes y los despegues de los aviones, una pre-departure sequence. Los demás pasos de implementación no los hemos entendido como básicos e imprescindibles para el simulador y no se han implementado.

2.3.1 Information Sharing

Definido por Eurocontrol, es el encargado de administrar la información del ACDM. Se encarga de recibir toda la información que se va generando y de tenerla disponible para que puedan acceder a ella en cualquier momento.

Permitiendo así que todos los actores tengan en todo momento toda la información y puedan actuar coordinada y eficientemente. Y es fundamental para poder llegar a implementar los siguientes pasos de implementación.

En la figura 2-3 del Manual de Eurocontrol podemos ver mejor la relevancia que tiene el Inforamation Sharing dentro del sistema.

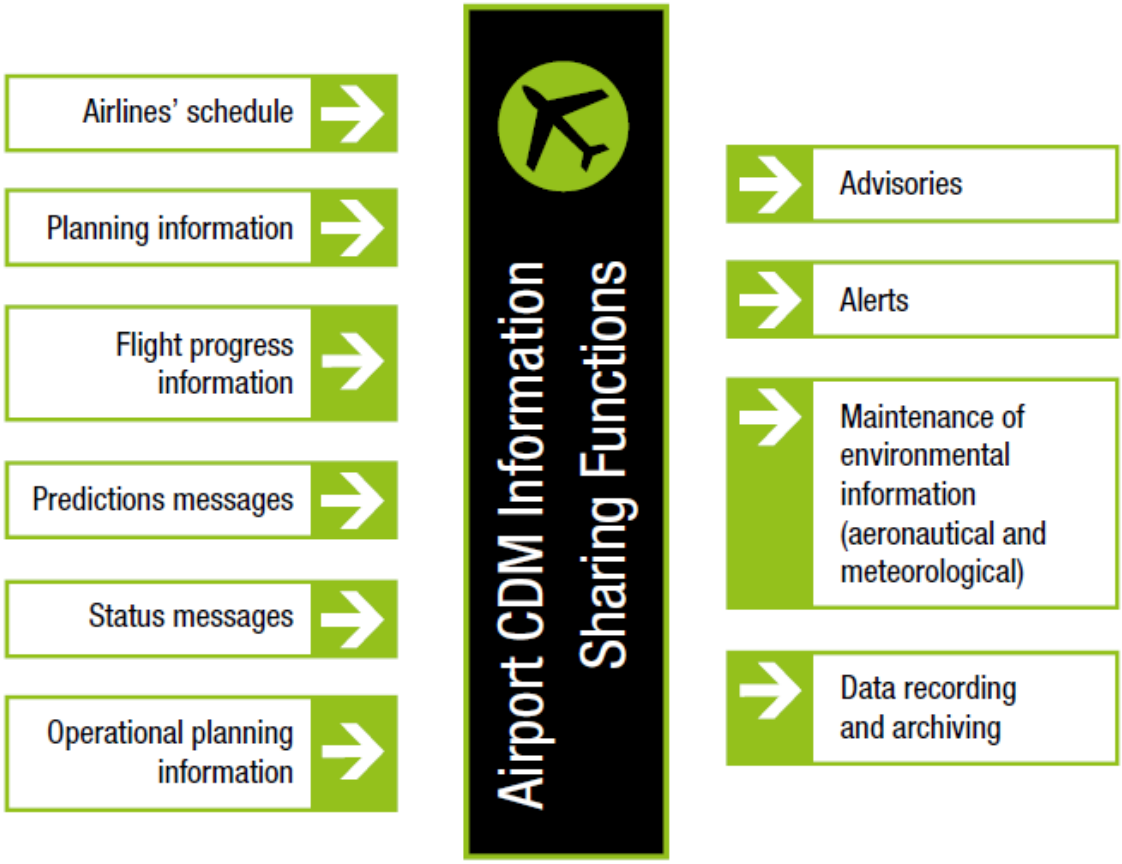


Figura 2-3: Servicios Information Sharing [4]

Information Sharing facilita la toma de decisiones a nivel local de cada uno de los actores y la aplicación de los diferentes procesos ACDM a través de:

- Conecta los sistemas de procesamiento de datos de todos los *stakeholders*.
- Proporcionar un conjunto único y común de datos que describen el estado y las intenciones de un vuelo.
- Servir de plataforma para el intercambio de información entre socios

Para dar estos servicios, el Information sharing:

- Coteja y distribuye la planificación y vuelo información de progreso, procedentes de ATFM (CFMU), ATC, las aerolíneas, los operadores del aeropuerto y las empresas de handling.
- Coteja y distribuye las predicciones de eventos y mensajes de estado

- Genera avisos y alertas.
- Proporciona información sobre el estado de ayudas y sistemas aeronáuticos y fuentes meteorológicas.
- Permite la grabación y archivo de datos para análisis estadísticos y otros estudios (por ejemplo, con fines financieros).

2.3.1.1 Implementación Information Sharing

Para realizar esta implementación se han definido una serie de pasos o funcionalidades que la plataforma Information Sharing tiene que cumplir para asegurar un correcto funcionamiento para el sistema ACDM [4].

Creación de una plataforma Information Sharing para ACDM.

1. Definir un formato estándar para la información que se transmitirá y se guardará.
2. Establecer un servicio de envío de información y datos automático, que distribuya la información al recibirla.
3. Relacionar los procedimientos, locales y generales, de los actores, directamente con esta plataforma. (Que los actores envíen la información a Information Sharing justo cuando realizan los procedimientos).
4. Definir mensajes de alerta de actualizaciones de información que se envíen al detectar que se realizan nuevos eventos y cálculos.
5. Utilización de una interfaz que muestre todos los datos e información que contiene el Information Sharing.
6. Ya que estos son requisitos obligatorios que tiene que cumplir la plataforma en un ACDM real, el agente que representa al Information Sharing en el programa los cumple todos correcta y eficientemente.

2.3.1.2 Plataforma Information Sharing

Una de las partes más importantes de esta plataforma es la definición de una estructura clara y eficiente de la información, para ayudar a que todos los actores entiendan y puedan funcionar con esta información.

Definir un formato de los datos es esencial para evitar inconsistencias o problemas de reconocimiento de datos. Como los diferentes *stakeholders* tienen diferentes formatos; se deben especificar y desarrollar filtros y conversores con el fin de interconectar los diferentes sistemas y evitar problemas de datos.
Una ejecución eficiente requiere que los estándares utilizados, incluidos los convenios de datos, se aceptada y acatada por todos.

Los estándares utilizados también deberán cumplir los requisitos de seguridad, fiabilidad y eficiencia, sin crear una situación de exceso y el aumento de los costos innecesariamente. Es necesario que se llegue a un acuerdo entre todos los asociados, respecto al formato de los datos y convenciones de los mensajes que se intercambiarán.

Estructura de datos en el programa

La estructura de datos que se ha definido en este programa se ha basado en las necesidades que tendrá el ACDM durante una simulación, es decir, que

información sobre los diferentes aviones es necesaria para los diferentes cálculos y procesos que se harán.

En concreto, y para definir cuáles son los tiempos del avión que se necesitarán, hemos hecho un análisis de la división del vuelo en Milestones (Milestone approach, explicada más adelante en este apartado).
De esta manera, hemos obtenido que la estructura de datos que se seguirá en el programa para enviar y guardar los datos de los aviones tendrá estas variables:

- ID – Muestra el callsign del avión
- Status – Describe el momento del vuelo en el que se encuentra el avión
- Milestone – Muestra el milestone en el que se encuentra el avión
- Tiempos – Muestra los valores de los tiempos principales del avión.
- Tiempos secundarios – Muestra los valores de los tiempos secundarios.
- Tiempo vuelo – Muestra los valores de los tiempos en vuelo.
- Incidencias – Muestra los retrasos que se han producido en la simulación. Describiendo que agente realizó el retraso, que tiempo se retrasa y los tiempos.
- ADEP – Muestra cual es el aeropuerto de procedencia del vuelo
- ADES – Muestra cual es el aeropuerto de destino del vuelo
- Modelo – Muestra cual es el modelo del avión
- Categoría – Muestra a que categoría pertenece el modelo del avión
- AO – Muestra la aerolínea que opera el avión
- Matrícula – Se utiliza para guardar el segundo callsign del avión en caso de que se dé el caso de que aterrice y despegue del aeropuerto el mismo día, en esos casos tiene un callsign para cada vuelo.

Las actualizaciones de esta información se pueden seguir en el programa, tanto a través de la interfaz gráfica como por la consola. En la figura 2-4 vemos como el programa muestra las actualizaciones de los parámetros relevantes de cada avión.

```
IS: BroadCastInformation
Aircraft: IBE220
  STATUS: LANDED
  Times:
        ETOT' = 930
        ELDT = 1045
        ETOT = 1155
        EIBT = 1100
        EOBT = 1135
  Milestone: 6
  Secondary times:
     ACGT: 1103
     ASBT: 1110
     ARDT: 1128
     ASRT: 1129
     TSAT: 1130
Warnings:
EOBT delayed from 1125 by 1135 by GH
ASBT delayed from 1100 by 1110 by AO-IBE
ARDT delayed from 1110 by 1128 by AO-IBE
ASRT delayed from 1119 by 1129 by AO-IBE
```

Figura 2-4: Ejemplo actualización avión por consola

Estos tiempos que se muestran coinciden con los tiempos que se necesitan para el cálculo de los diferentes Milestones del ciclo CDM de un avión. Por tanto, son los tiempos que el sistema necesita para realizar los procesos ACDM. Estos tiempos son los que encontramos en la tabla 2-1:

Tiempos principales	**ETOT'**	Tiempo estimado de despegue en aeropuerto de origen
	ELDT	Tiempo estimado de aterrizaje
	ETOT	Tiempo estimado de despegue en aeropuerto simulado
	EIBT	Tiempo estimado de llegada a Gate
	EOBT	Tiempo estimado de salida de Gate
Tiempos secundarios	**ACGT**	Tiempo de inicio de handling
	ASBT	Tiempo inicio embarque en el avión
	ARDT	Tiempo en el que el avión estará listo para empezar el vuelo
	ASRT	Tiempo en el que se solicita encender motores
	TSAT	Tiempo en el que se aprueba encender motores
Tiempos de vuelo	**TFIR**	Tiempo en el que el avión entra en la FIR del aeropuerto
	TAPP	Tiempo en el que el avión empieza la aproximación final para aterrizar

Tabla 2-1: Tiempos en los datos de un avión

El parámetro "Status" nos sirve para saber rápidamente en qué fase del ciclo ACDM (en qué milestone) se encuentra el avión en ese momento. La lista de status es limitada y, para que crear un formato predeterminado y que todos los actores lo entiendan rápidamente, Eurocontrol también define una lista de qué status se utilizarán en el ACDM, figura 2-5.

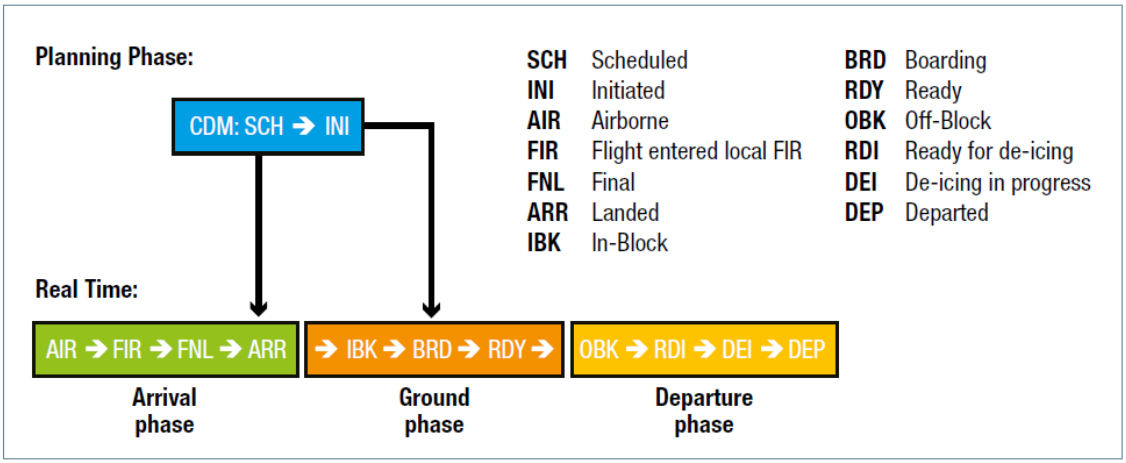


Figura 2-5: Lista Status ACDM [4]

En la información que se utiliza en un ACDM real podremos encontrar más datos, como información sobre que Gate se ha asignado al avión, que procedimiento ha seguido para aterrizar, etc. Estos datos no son relevantes en el programa tal y como está actualmente, ya que no se refieren a características del aeropuerto que no están implementadas (por ejemplo no se ha implementado un agente que simula la asignación de gates de los aviones que llegan al aeropuerto), y por eso no están definidos en el programa.

Todos estos datos llegan al Information Sharing a través de diferentes actores, ya que cada uno tiene información del avión en diferentes fases. El manual también define la fuente da cada una de estas informaciones, y muestra un esquema donde podemos ver la información que cada actor "sube" a la plataforma. Este esquema se puede ver en la figura 2-6 que tenemos a continuación.

En el caso del simulador que hemos implementado, la información que provee cada uno de los actores la podremos ver más adelante en la memoria, de una manera más detallada que en la figura 2-6, ya que veremos qué tiempos calcula cada agente y cuando.

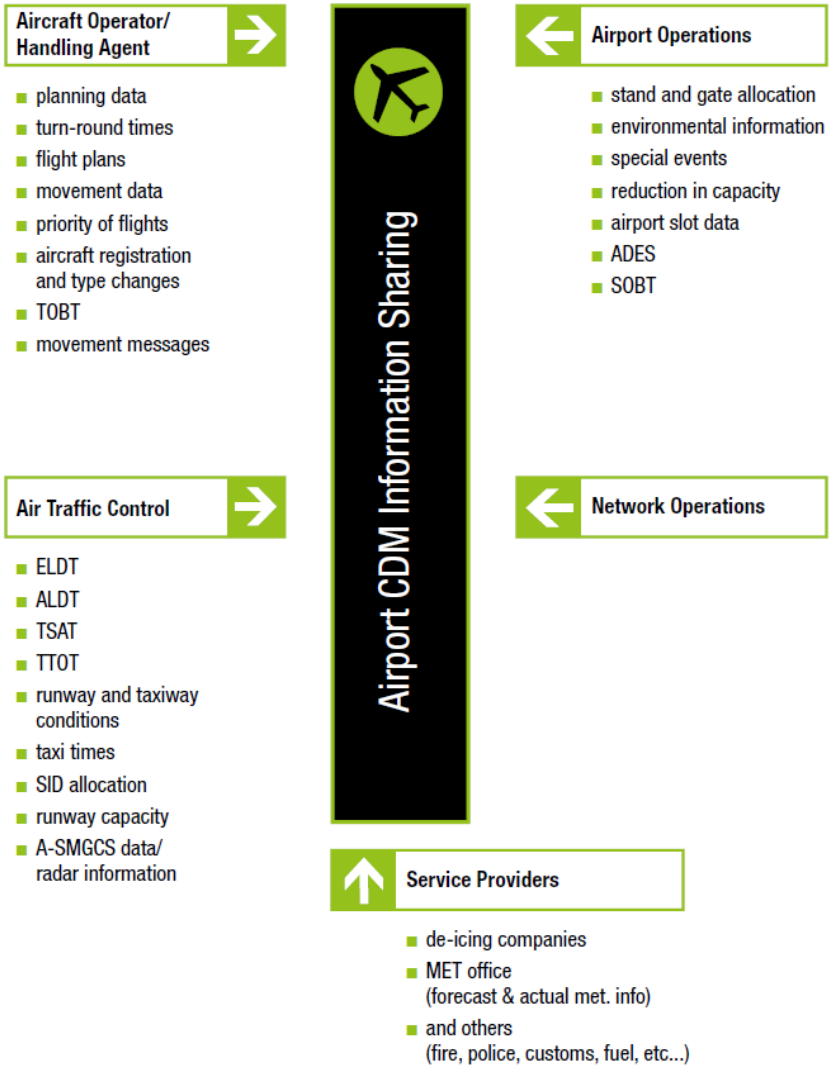


Figura 2-6: Esquema información actores [4]

Interfaz del Information Sharing

Debido a que en el programa se ha hecho una versión “académica” de esta plataforma, nos hemos centrado en definir correctamente la estructura de la información y conseguir que realice todos los cálculos y envío de información y alertas que encontramos en la vida real. Pero a la hora de mostrar la interfaz, la hemos adecuado a que muestre la información que más nos interesa para hacer los diferentes análisis, y por eso podemos ver como la interfaz sugerida por Eurocontrol [4] y la que podemos encontrar en el programa son diferentes en algunos puntos.

Las diferencias entre la interfaz propuesta por Eurocontrol y la utilizada finalmente en el programa se basan en la información que cada una quiere mostrar.

En la figura 2-7 vemos como la interfaz propuesta muestra información sobre la situación general en el aeropuerto, pocas columnas que muestren tiempos de los aviones, e información que en el programa no se utiliza, como la SID utilizada por cada avión o la situación del handling.

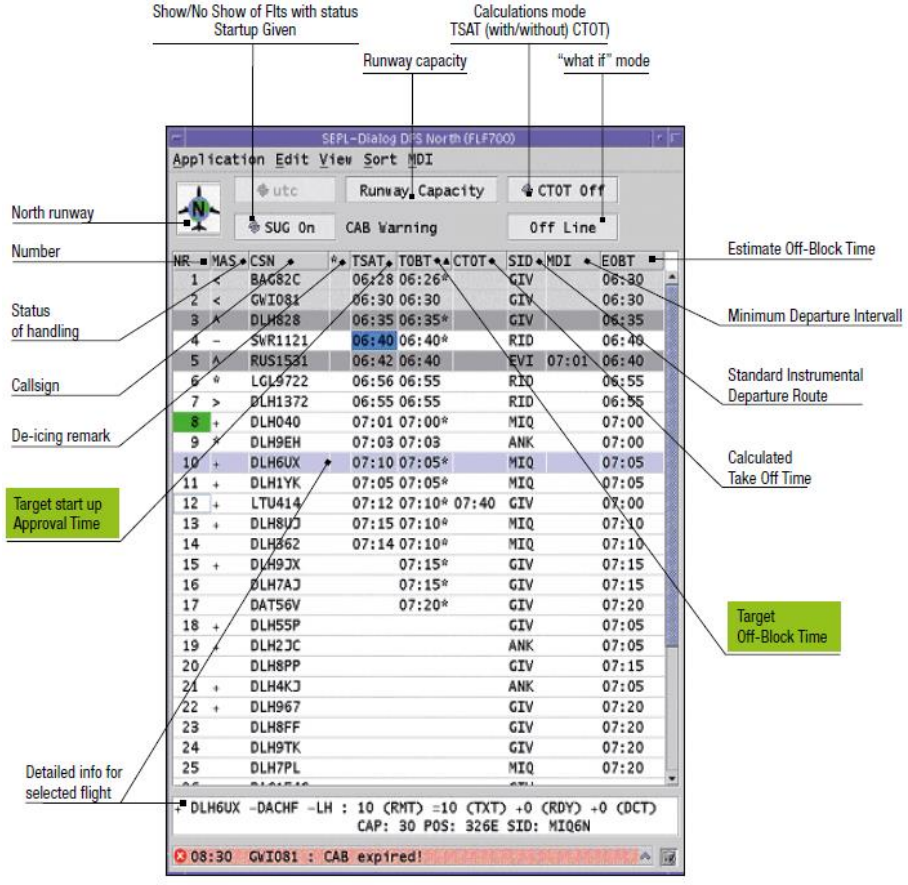


Figura 2-7: Interfaz propuesta por Eurocontrol [4]

En cambio, podemos ver como en la interfaz que finalmente se ha utilizado en el programa, figura 2-8, se muestran los parámetros de cada avión que son importantes para entender y analizar el tráfico aéreo que se está calculando.

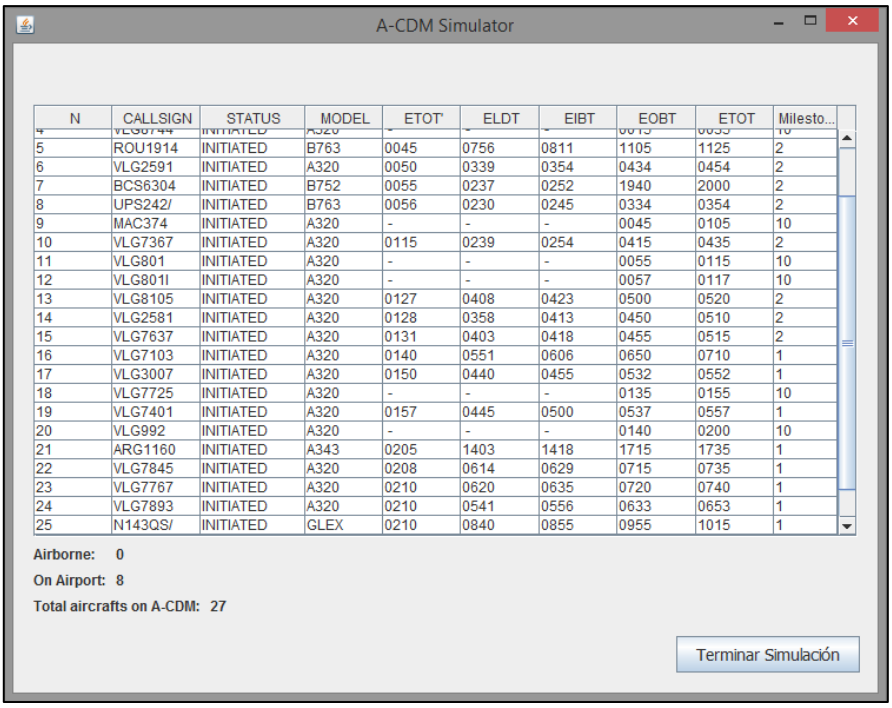


Figura 2-8: Interfaz IS en el programa

2.3.2 Milestone Aproach

Se realiza un seguimiento de cada uno de los vuelos a través de una secuencia de eventos, conocidos como milestones, y la realización de diferentes cálculos y previsiones de tiempos a través de estos. Los diferentes *stakeholders* pueden ser responsables de diferentes milestones, con el objetivo de que la información de la evolución del vuelo sea común para todos.

El objetivo principal de este enfoque a través de milestones es mejorar aún más el conocimiento de la situación común de todos los actores. Más específicamente, los objetivos son los siguientes:

- Determinar eventos significativos con el fin de realizar el seguimiento del el progreso de los vuelos y la distribución de éstos eventos clave como Milestones
- Definir las actualizaciones y los factores desencadenantes de la información: nuevos parámetros, estimaciones de actualizaciones posteriores, mensajes de alerta, notificaciones, etc.
- Definir la calidad de los datos en términos de precisión, fiabilidad, estabilidad y previsibilidad, basado en una ventana de tiempo en movimiento.
- Habilitar la toma de decisiones con antelación cuando suceden imprevistos en un evento.
- Mejorar la calidad de la información.

De esta manera, sabemos que esta *Milestone Approach* se centra en:

- Un conjunto de eventos o milestones seleccionados a lo largo del progreso del vuelo (llegada, aterrizaje, taxi-in, turn around, taxi-out y salida), donde los socios implicados en el cambio en el proceso de vuelo.
- El cálculo de los diferentes tiempos de los eventos del avión a partir del conocimiento de los milestones previos.

La información del vuelo se calcula mediante los milestones que ya se han producido, por lo que el rendimiento de tiempo entre cada milestone y su actualización en tiempo real se convierte en un parámetro fundamenta. Que esta información sea compartida en tiempo real es de suma importancia para que cada *stakeholder* pueda reaccionar a tiempo para actualizar sus milestones y realizar sus cálculos, de esta manera se mejora el rendimiento en la previsibilidad y la eficiencia del global.

Estos cálculos a través de milestones ya producidos los podemos ver, por ejemplo, en una vez que el vuelo se ha activado en el A-CDM, cuando CFMU procesa este plan de vuelo, se pueden utilizar varios valores por defecto basados en reglas genéricas y locales para definir, por ejemplo, los tiempos de taxi, diferentes tiempos de handling y un Target Take Off Time. Otro ejemplo lo encontramos en los operadores de aeronaves y los controladores se puede revisar y actualizar algunos valores con el fin de mejorar la calidad de la predicción, basándose en el número de pasajeros o el progreso de vuelos de conexión.

Una de los objetivos principales y por el que se puede definir el A-CDM como útil es la predicción y cálculo del TOBT (Target Off-Block Time). Su calidad se puede evaluar, midiendo su puntualidad, precisión y previsibilidad, y es un buen indicativo de cuan buena ha sido la implementación del A-CDM.

La confianza en el A-CDM se basa en la calidad de la TOBT, que a su vez depende de otros milestones, por lo que la precisión de cada milestone también debe ser analizada para identificar que partes necesitan ser mejoradas para obtener TOBT con precisión.

El progreso del vuelo se controla de forma automática y progresa a través de la ejecución de cada uno de los milestones; Se va añadiendo más información y se modifica a medida que esta esté disponible, así como los milestones se actualizan en consecuencia a las actualizaciones de la información.

2.3.2.1 **Milestones definidos**

Un total de 16 milestones básicos se han definido. La lista de milestones es indicativa, algunos aeropuertos pueden necesitar incluido otros para cubrir las actualizaciones adicionales de información sobre eventos clave, como los procesos de de-icing después de aterrizar. Debido a que encontramos algunos milestones que definen procedimientos locales, los podemos definir como recomendados y no obligatorios.

En la tabla 2-2 podemos ver la lista de estos milestones, con la información de a que evento del vuelo corresponden y el tiempo al que corresponden (con el nombre que se ha utilizado en el programa).

Milestone	Definición	Tiempo
1	Activación Flight Plan	3h antes de ETOT’
2	2h antes ETOT’	2h antes de ETOT’
3	Despegue en origen	ETOT’
4	Entrada en FIR	TFIR

5	Inicio Aproximación Final	TAPP
6	Aterrizaje	ELDT
7	Llegada a Gate (In-Block)	EIBT
8	Inicio Ground Handling	ACGT
9	Confirmación final de TOBT	Depende del aeropuerto
10	Definición de TSAT	Depende del aeropuerto
11	Inicio Embarque	ASBT
12	Aircraft Ready	ARDT
13	Petición encender motores	ASRT
14	Aprobación encender motores	TSAT
15	Inicio taxi de salida (Out-Block)	EOBT
16	Despegue	ETOT

Tabla 2-2: Milestones definidos

Estos milestones, además, se pueden diferenciar en 3 grupos, según a que en parte del vuelo se producen, como son considerados por nuestro aeropuerto: Trafico de llegada (INBOUND), tráfico en el aeropuerto (TURN AROUND) o tráfico de salida (OUTBOUND). Esta clasificación se puede ver de una manera más gráfica en la figura 2-9:

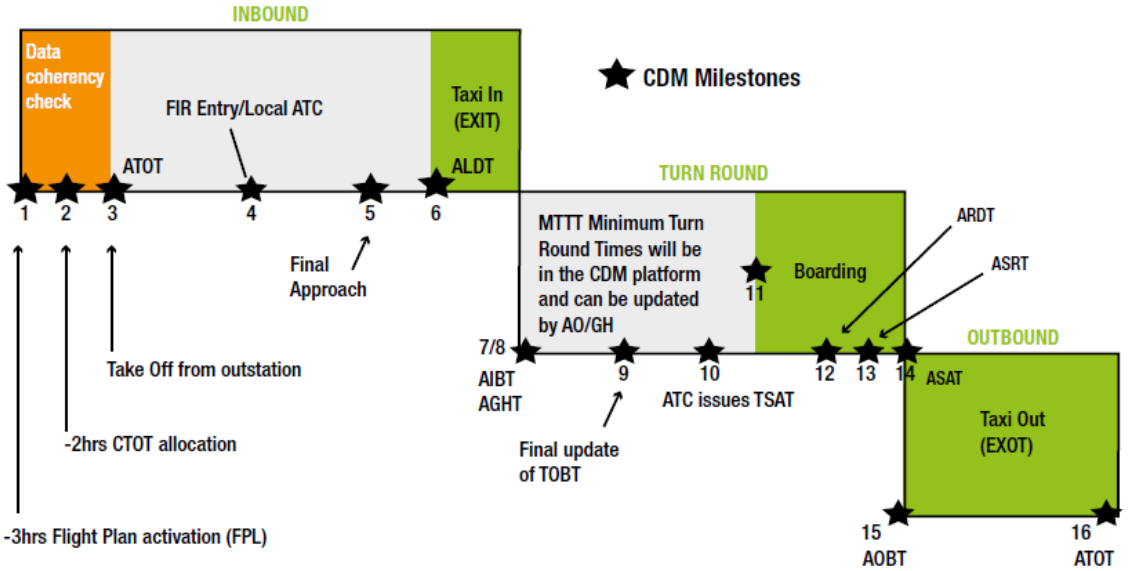


Figura 2-9: Esquema Milestones [4]

Estos milestones los podemos encontrar analizados en el apéndice A de la memoria, donde se ha realizado un análisis en detalle para poder determinar:

- Cuando se ejecuta cada milestone.
- Que actor detecta e inicia cada milestone.
- Que actores realizan cálculos de tiempos a partir de la información actualizada.
- Que tiempos se calculan/definen/actualizan en cada milestone

CAPITULO 3. Descripción del Proyecto

En este capítulo intentamos responder a la siguiente pregunta: ¿Cómo se ha planteado la implementación del A-CDM Simulation Framework? Es decir, buscamos fijar las bases de la aplicación y los requisitos.

En el desarrollo de un software comercial estas bases y requisitos las pondría el cliente, pero en este caso se sacan a partir de la idea inicial de lo que sería el proyecto y, debido a que se ha seguido un modelo de software incremental, se han ido añadiendo nuevos requisitos y funcionalidades mientras ha ido durando el desarrollo del simulador.

Esto quiere decir que una vez sentadas las bases del programa, se han ido añadiendo funcionalidades específicas no previstas; como por ejemplo el hecho de añadir que ATC realice una pre-departure sequence de los aviones en plataforma, y así optimizar los tiempos de taxi de salida.

Este proyecto ha tenido una duración de casi 1 año con ambos trabajando a la vez en la programación y el desarrollo. Se han desarrollado diferentes versiones, cada vez más optimizadas y con funcionalidades añadidas, de las cuales se hablará más adelante en esta memoria.

Es importante comentar que la primera la primera parte de este proyecto fue publicada y seleccionada para ser presentada en el Congreso Internacional de Investigación en Transporte Aéreo (ICRAT), que se celebró en mayo del 2014 en Estambul (Turquía), donde el alumno Ivan Garcia presento el proyecto. El artículo publicado fue escrito por los profesores de la UPC Miguel Valero y Xavier Prats, el profesor de la Universidad de Westminister Luís Delgado, y el propio Ivan Garcia [3]

3.1 *Desarrollo del proyecto*

A la hora de realizar este proyecto, una buena organización y trabajo previos eran importantes debido a que no estábamos familiarizados con las herramientas empleadas a lo largo del proyecto, Java y JADE. (En el caso de JADE se tuvo que hacer más trabajo de aprendizaje autónomo dado que no había profesores en la escuela que hubieran trabajado previamente con él)
Así pues en el desarrollo del software se han definido 5 fases diferenciadas:

- Planteamiento, definición y creación de los Agentes (y comunicación entre ellos)
- Implementación del Airport Collaborative Decision Making para un único avión (Primeros milestones)
- Implementación del Airport Collaborative Decision Making para un único avión (16 milestones)
- Implementación del Airport Collaborative Decision Making para un tráfico real (multi-avión)
- Implementación de las funcionalidades extra y optimización del A-CDM

3.2 *Fases del desarrollo*

Originalmente se pensó en implementar el ACDM para n-aviones, con lectura de datos más simples y sin tráfico aéreo (definido por nosotros).
Pero al observar que la utilidad de la aplicación sería bastante baja al no utilizar tráficos realistas, se optó por realizarla con tráfico real e implementar diferentes funcionalidades para mejorar el realismo del comportamiento y de los resultados.

Finalmente, dada la complejidad de la aplicación, se optó por realizar una versión inicial, en la que familiarizarnos con los sistemas multiagentes, las comunicaciones y asentar las bases de las comunicaciones del programa. Utilizando como conductor el programar el ciclo A-CDM de un único avión hasta el milestone donde aterriza en el aeropuerto estudiado (Milestone 6). En esta fase se definieron las herramientas necesarias para el desarrollo del software (un sistema multiagente), y se realizó un planteamiento básico de este (agentes necesarios en nuestro sistema y funciones básicas de estos). Es por este motivo que hemos considerado más oportuno incluir esta fase dentro de la descripción del proyecto y no en el desarrollo del software.

También es necesario decir que, durante esta fase, también se llevó a cabo la ejecución de diferentes agentes y se comprobó de qué manera se establecía la comunicación entre ellos.

Entre esta primera fase y la definitiva con tráfico aéreo real, se realizó una fase intermedia para optimizar la definición del ciclo A-CDM en el programa, se buscó dejar bien definidos y programados todos los comportamientos que suceden durante los 16 Milestones que sigue un avión en todo el ciclo, implementación que se conservaría para la fase final.

Por otra parte, y debido a que en esta versión el programa era totalmente determinista y a pequeña escala (siguiendo el recorrido de un avión) se podía adivinar fácilmente el resultado, se ha realizado una última fase de implementación de mejoras extra, no previstas durante la definición del proyecto, para hacer más realista el comportamiento del simulador. Estas mejoras se centran en simular diferentes comportamientos de los agentes del programa. Estas mejoras se detallan como funcionalidades extra porque aparte de mejorar el realismo del programa, añaden posibilidades a la simulación que en las versiones anteriores no encontramos.

Estas funcionalidades extra se explicarán con más detalle en próximos apartados de esta memoria, pero consideramos necesario enumerarlas y hacer un pequeño comentario para dar una idea de en qué consistirán:

- Definición retrasos: Retrasos imprevistos no presentes en el tráfico aéreo utilizado como input, ejecutados en cualquier momento de la simulación. Se definen a través de una interfaz en la que puedes ver todo el conjunto de aviones, del tráfico aéreo que se utilizará en la simulación, y seleccionar que aviones retrasar, cuanto tiempo y en qué momento de su ciclo A-CDM.

Útiles porque añaden realismo a la simulación. También se usan para analizar el comportamiento del aeropuerto a diferentes situaciones, y observar a que aviones afectaría y la propagación y el volumen de minutos de retraso que aparecerían. Por ejemplo, se puede simular que hay una tormenta en la FIR en una ventana de tiempo determinada y ver cómo afecta al tráfico entrante.

- Pre-departure sequence: (Llamada así debido al manual de Eurocontrol [4], pero está programada para departures y arrivals). A la hora de regular las salidas y las llegadas al aeropuerto, los ATC tienen que tener en cuenta diferentes aspectos como la capacidad de la pista en ese instante y la categoría de los aviones que van a aterrizar/despegar. Esto es debido a que la distancia recomendada entre dos despegues o aterrizajes depende de la categoría de los aviones que van a realizar estas maniobras, por ejemplo, hay que dejar mayor distancia entre un categoría Heavy y un Light que entre dos Light.

  Además de este tiempo variable, esta funcionalidad también ofrece la posibilidad de utilizar dos políticas diferentes a la hora de manejar un avión que ha sido retrasado, ofreciendo así la posibilidad de comparar los resultados con cada una de ellas. Estas políticas se basan en de qué manera encontramos un hueco en la cola de aviones de la pre-departure sequence para reubicar un avión que retrasa su salida.

- Reducción capacidad de pista: Frente a una situación adversa que obliga a reducir el número de operaciones hora que puede manejar las pistas del aeropuerto, se ha simulado el comportamiento de los ATC del aeropuerto para garantizar la seguridad de aviones y pasajeros.

  ATC procede a espaciar los aviones de manera que el número de operaciones a la hora que sucedan sea el óptimo para el funcionamiento del aeropuerto en esa determinada situación. Esta funcionalidad que añade retrasos a los aviones necesarios y propaga estos retrasos para garantizar la seguridad, está ligada a las dos funcionalidades anteriores. Las ventanas de tiempo donde tendremos capacidad de pista reducida también se pueden programar a través de una interfaz al inicio del programa.

- Retrasos aleatorios: Para evitar que el programa tenga un comportamiento determinista y añadir aleatoriedad a los resultados, se ha añadido la opción de que los tiempos varíen con una distribución estadística normal. Esta variación de los tiempos se lleva a cabo en el

momento que ocurre el milestone del tiempo en cuestión, intentando simular el caso en que por cualquier cosa el tiempo estimado varia levemente del tiempo real.

Finalmente se optó por también añadir funciones de guardar/cargar, tanto resultados como condiciones iniciales, para añadir repetitividad a las simulaciones y poder hacer análisis con más precisión. Por tanto es posible cargar directamente las condiciones iniciales (tráfico aéreo, retrasos y los diferentes parámetros del aeropuerto y la simulación) sin tener que volverlas a introducir, pudiendo cambiar cualquiera de ellas. Además, si ya tenemos unos resultados previos se pueden cargar directamente, sin tener que volver a simular, para analizarlos con la interfaz de análisis; también se ha añadido la opción de cargar dos sets de resultados para poder analizar conjuntamente y compararlos.

3.3 *Requisitos del Software*

La extracción de requisitos es un parte fundamental en el desarrollo de cualquier software. Una buena base de requisitos proporciona una amplia serie de requisitos, pero los que más nos interesan en este proyecto son:

- Establecer un acuerdo cliente – desarrolladores en relación al producto que se va realizar. En nuestro caso hacemos el papel de desarrolladores y de clientes, pero es necesario fijar las bases de lo que se pretende desarrollar.
- Proporcionar una base de referencia para una futura mejora. Definiendo claramente las prestaciones de nuestro programa y asegurando una fase de validación y verificación de estas, creamos una base a partir de la cual se puede continuar el programa en un futuro.
- Proporcionar una referencia para la validación y verificación del software. Validando y verificando los requisitos establecidos nos aseguramos de cumplir con las especificaciones que el cliente desea.

En muchos proyectos de software se tiene en cuenta distinguir entre dos tipos de usuarios “usuario cliente” y “administrador”; en nuestro caso no se hace tal distinción ya que este software está pensado para ser continuado por otros alumnos/investigadores y no se espera que se utilice, tal y como esta, para aplicaciones reales y análisis para aeropuertos.

Para esquematizar este análisis de requisitos y poder implementarlos mejor después en el software, se ha hecho una distinción de estos en tres grandes grupos: Requisitos de diseño, Requisitos funcionales y Requisitos de interfaz.

También es importante comentar que este proyecto ha ido evolucionando con el tiempo, ya que al principio se tenía una idea de hacer un simulador mucho más

básico, con un tráfico aéreo no real, con menos trabajo en el realismo de los comportamientos y sin las funcionalidades extra explicadas anteriormente. Esto hace que esta lista de requisitos haya tenido que ir evolucionando con el proyecto. La lista plasmada a continuación se corresponde a la última versión, a como ha quedado el programa final, y por tanto es más extensa que las anteriores versiones.

3.3.1 Requisitos de diseño

- Implementar un software que sirva como entorno de simulación para la ejecución de simulaciones multiagente.
- Implementar un simulador de Airport Collaborative Decision Making.
- Implementar un simulador A-CDM que simule con datos reales de la base de datos DDR2 de Eurocontrol.
- El programa deberá implementar los agentes reales que aparecen en el A-CDM: CFMU, ATC, aerolíneas, empresas de *Ground Handlig* y el centro de control del aeropuerto.
- Implementar comportamientos realistas en cada agente para el manejo de la información de los aviones durante los 16 milestones del ciclo A-CDM.
- El programa deberá implementar un sistema de comunicaciones entre agentes CDM centralizado, a través de un agente que reciba y comunique la información de los aviones (seguir indicaciones Information Sharing de Eurocontrol [4].
- El programa deberá de ser capaz de guardar y mostrar los resultados de la simulación.
- Implementar la posibilidad de introducir retrasos o incidentes en la simulación.
- Implementar la posibilidad de añadir aleatoriedad a la simulación para añadir realismo.
- Debe implementarse un agente externo que sirva de ejemplo para futuros programadores.

3.3.2 Requisitos funcionales

- La aplicación debe estar programada en Java y poder ejecutarse desde Eclipse.
- El programa deberá usar el middleware JADE para la implementación de los agentes.
- Deberá darse la posibilidad de capturar las comunicaciones entre agentes realizadas durante una simulación.
- Implementar un agente que centralice las comunicaciones de los agentes CDM y se encargue de recibir, registrar e informar de estas.

- Implementar un tipo de mensaje que asegure una comunicación fluida y un procesamiento rápido por parte de los agentes.
- Implementar un objeto que represente todos los parámetros y tiempos de un avión que sean necesarios para la simulación del A-CDM.
- Implementar una tabla para guardar y administrar el tráfico aéreo en cada uno de los agentes que forman parte de la simulación.
- Asegurar que todas las tablas de los agentes de la simulación tienen, en todo momento, la misma información de las actualizaciones del tráfico aéreo.
- Implementar un agente que maneje la simulación automática, iniciando cada uno de los milestones que se producen en el tráfico aéreo.
- Implementar un agente que controle los parámetros iniciales de la simulación y permita modificarlos.
- Implementar un agente que se encargue de analizar el archivo de la base de datos DDR2 utilizando los parámetros definidos, y que genere el archivo de tráfico aéreo que utilizará el programa.
- Implementar un agente que genere todos los documentos Input necesarios a partir del documento de tráfico aéreo del agente DDR2.
- El programa deberá permitir la repetitividad de las simulaciones generando archivos que guarden los parámetros iniciales y los retrasos de la simulación.
- El programa deberá generar un documento con los resultados de la simulación del tráfico aéreo.
- El programa deberá permitir que se pueda comprobar si la ejecución de la simulación ha sido correcta. Generando un documento con los milestones ejecutados y en orden de ejecución.
- El programa deberá permitir el fácil acceso a cambiar las diferentes constantes utilizadas para los cálculos de tiempos.
- Los agentes que se encargan de la simulación del CDM deberán actuar de manera realista, actuando cada uno sobre los tiempos del avión que toca según el stakeholder que representan en el programa.
- El programa debe asegurar la correcta ejecución de los milestones de la simulación ordenándolos todos cronológicamente en cada actualización.
- El programa debe terminar la simulación e iniciar la fase de guardado y muestra de los resultados automáticamente, al acabarse los milestones de la simulación.
- El programa debe soportar y simular retrasos e incidentes en cualquiera de los aviones del tráfico aéreo en cualquier momento de la simulación.
- Implementar un agente que se encargue de cargar y administrar los retrasos. Que permita crear nuevos y borrar los definidos.

- Implementar un agente que se encargue de simular un proceso de pre-departure sequence. Que organice las salidas y llegadas de los aviones en función de sus tiempos, para cumplir con capacidad definida de las pistas.
- Implementar un agente que cargue y permita el análisis de los resultados de la simulación.
- El programa debe poder generar archivos con los resultados de los análisis de los resultados de la simulación. Mostrando diferentes parámetros de este análisis y extrayendo los datos de las gráficas generadas para que se puedan utilizar en otro programa.
- El programa debe poder cargar otro set de resultados de otra simulación para permitir al usuario hacer comparar resultados y análisis.
- El programa debe dar la opción de saltarse la simulación e ir directamente a analizar unos resultados obtenidos en otra simulación.

3.3.3 Requisitos de interfaz

- El programa debe dar al usuario la posibilidad de cargar los archivos input de la simulación a través de escribir la ruta de estos.
- El programa debe permitir cambiar los parámetros iniciales de la simulación que no se encuentren en el archivo.
- El programa debe permitir al usuario administrar los retrasos de la simulación gráficamente, a partir de cargar un archivo y crear/borrar retrasos.
- El programa debe de ofrecer una interfaz que facilite al usuario la creación de retrasos en el formato que los simulará el programa. Asegurando que dichos retrasos se ejecutarán sin problemas y están definidos correctamente.
- El programa debe ofrecer la posibilidad de decidir el tiempo de simulación (tiempo entre la ejecución de cada milestone) y avisar del tiempo total que tardará en simular.
- El programa debe mostrara la simulación en tiempo real. Mostrando los aviones definidos en el A-CDM, sus tiempos principales y en qué momento (milestone) de su ciclo CDM están.
- El programa debe ofrecer la posibilidad de parar la simulación en cualquier momento a través de un botón en la interfaz.
- El programa debe ofrecer la posibilidad de no simular y pasar directamente al apartado de mostrar y analizar resultados, dando la posibilidad de cargar un archivo de resultados a través de definir la ruta en la que se encuentra.

- El programa debe mostrar los resultados de la simulación en una interfaz. Mostrando los mismos tiempos y parámetros que la interfaz que muestra la simulación.
- El programa debe ofrecer la posibilidad de analizar los resultados (cargados u obtenidos). Mostrando gráficas de la evolución de los retrasos en el tiempo de un parámetro elegido por el usuario, además de diferentes parámetros de ese análisis, como medias, máximos y mínimos.

3.4 *Sistema MultiAgente*

Se considera un sistema multiagente (MAS) aquel sistema compuesto por la interacción de múltiples agentes. Un sistema multiagente se suele usar para resolver problemas difíciles o imposibles de resolver para un solo agente (o sistema), ya que sólo posee una parte de la información necesaria para la resolución de dicho problema.

Los sistemas multiagente son caracterizados principalmente porque cada agente tiene una información, habilidad o capacidad incompleta para solucionar el problema, es decir están limitados. Además existe una interacción entre ellos, no existe un sistema global de control, los datos están descentralizados y la computación es asíncrona.

Aunque no existe una definición comúnmente aceptada se considera un agente cualquier entidad, o proceso computacional, capaz de interactuar de forma robusta y flexible en un determinado entorno. Los agentes son también conocidos como agentes inteligentes y están incluidos en el campo de la inteligencia artificial. [7]

Las características fundamentales que deben poseer los agentes son:

- Autonomía: No necesita la intervención del usuario o de otros agentes
- Sociabilidad: Capacidad de interaccionar con otros agentes, a través de algún medio de comunicación. En este caso intercambio de mensajes.
- Reactividad: Agente inmerso en un entorno controlado, del que recibe estímulos (mensajes) y ante los que reacciona de una forma adecuada.
- Iniciativa: Reacciona a estímulos y cambios, actúa y toma la iniciativa para satisfacer sus objetivos.

Este tipo de sistema es el que más se adapta al proyecto que deseamos realizar puesto que no hemos implementado un sistema con diversos agentes debido a la imposibilidad de resolver un problema complejo, sino que lo hemos hecho para dotar de más realismo al comportamiento de los diferentes actores que participan en el A-CDM. Gracias a este sistema multiagente podremos observar el comportamiento individual de cada agente, su reacción a cambios y, lo que también nos interesa de cara a sacar conclusiones sobre el A-CDM real, las comunicaciones que se producen durante su ejecución.

Para implementar este modelo basado en agentes hemos empleado JADE (Java Agent Development Enviroment). JADE no solo nos servirá para la implementación de los diferentes agentes del A-CDM, sino que también nos permite la creación de agentes gráficos y la monitorización de las comunicaciones durante la ejecución.

3.4.1 JADE (Java Agent Development Enviroment)

JADE es un middleware (un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas) desarrollado por TILAB con el objetivo de propiciar el desarrollo de aplicaciones multiagente [8].

Además cumple con las especificaciones FIPA (Foundation for Intelligent Physical Agents). FIPA es una organización que se encarga de desarrollar especificaciones estándar para los sistemas basados en agente. De esta manera se busca la interoperabilidad entre los sistemas basados en agentes, independientemente de la implementación interna [8].

Por lo tanto, las características de JADE que más nos interesan son:

- Plataforma FIPA para la ejecución de los agentes.
- Una librería de clases para la programación de agentes. En nuestro programa los agentes heredan de la clase CDM, y esta a su vez de la clase *Agent* de *jade.core.Agent*. Esta clase nos proporciona comportamientos de interacción llamados “behaviours” [9]
- Mensajes entre agentes siguiendo los estándares FIPA ACL (Agent Communication Language).
- Programación en JAVA.

JADE también nos da la posibilidad de utilizar Ontologías para la definición de los agentes y todo el conjunto de comportamientos y mensajes que tendrá el sistema, gracias a estas ontologías en teoría es más sencillo que los diferentes agentes se entiendan. Se basa en la definición de los diferentes conceptos, mensajes y comportamientos en un solo archivo, del cual los agentes leerán para “hablar todos el mismo idioma”.

Este uso de las Ontologías no es seguido en este proyecto, debido al gran número de comportamientos diferentes, de agentes y a que no todos son agentes JADE (también encontramos interfaces gráficas que trabajan fuera de JADE). Se ha decidido no utilizar estas Ontologías porque se complicaban mucho debido al gran número de comportamientos y a que, debido a que el proyecto iba creciendo y se iban desarrollando nuevas funcionalidades, hubiera sido necesario cambiar estas ontologías muy a menudo.

La solución a no usar Ontologías se basa en utilizar la programación orientada a objetos de Java y preparar todo el programa para trabajar con objetos *AIRCRAFT* (explicado en los siguientes apartados). Esto quiere decir que se ha preparado a los agentes para hacer cálculos, tomar decisiones, enviar y recibir objetos *Aircraft*. Este objeto se podría definir como el objeto en el que se guarda toda la información que contiene un avión.

El hecho de trabajar con este objeto tiene mucha coherencia con la realidad de un A-CDM, donde se trabaja con los datos de un avión y tomando decisiones viendo como esto se actualizan en tiempo real.

JADE ha resultado ser una plataforma muy útil para la programación en agentes. Una vez acabado el proyecto podemos afirmar que ha sido de gran ayuda y nos ha simplificado mucho el trabajo, sobretodo en temas de definir las comunicaciones y capturar los mensajes enviados.

En esta memoria se ha hecho un pequeño tutorial de cómo empezar un proyecto con JADE. Este Apéndice “Métodos de cada Agente” explica cómo descargar e instalar el middleware, como configurarlo para poder ejecutarlo correctamente y como utilizar la interfaz que permite ver la simulación en tiempo real.
Aunque es importante comentar que el apartado de como ejecutar JADE se ha hecho con un ejemplo sencillo de programación con JADE, para ver cómo se tiene que configurar JADE para la ejecución de este programa hay que ir al Apéndice “Manual de usuario”.

Para aprender a programar para JADE recomendamos ir directamente a la web [9] o mirar el manual **[10]**.

3.4.2 Metodología GAIA

Para realizar la fase de diseño nos hemos ayudado de una metodología específica para el desarrollo de sistemas multiagente.

La metodología escogida, denominada metodología GAIA, nos permite establecer el diseño del sistema multiagente, obteniendo el número de agentes que se incluirán en la implementación final, las funciones que desempeñarán cada uno de ellos y las comunicaciones que se establecerán.

Esta metodología establece una serie de pasos para llegar a la fase de diseño. En primer lugar se debe realizar una fase de análisis dónde se definen los roles necesarios en nuestra aplicación, que se obtienen directamente de los requisitos que se han especificado, y la comunicación entre ellos (protocolos). Posteriormente, en la fase de diseño, se definen los agentes presentes en nuestro software y se asignan los roles para cada uno de ellos. Por último se establece la comunicación entre los agentes (servicios), que está directamente relacionada con la comunicación que se ha definido entre los roles, figura 3-1 **[11]**.

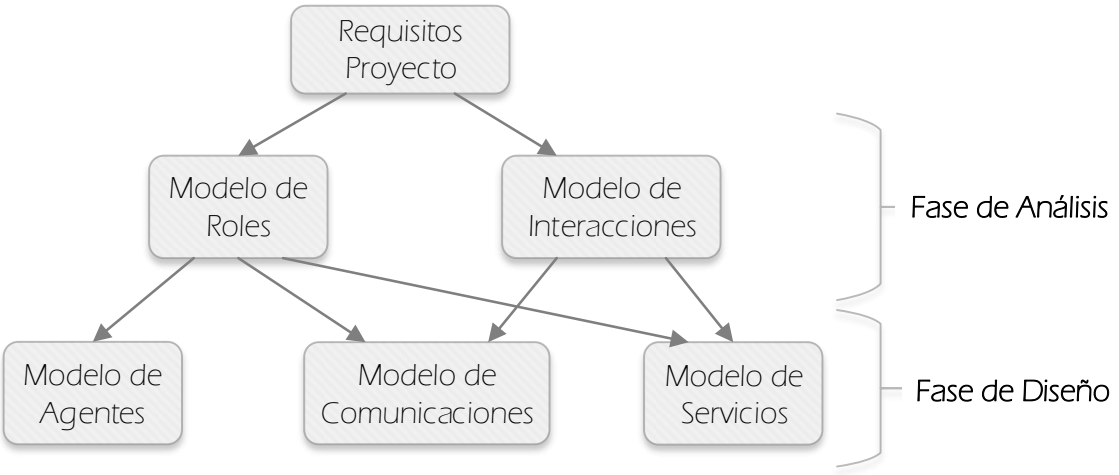


Figura 3-1: Fases Metodología GAIA

Esta metodología marca empezar con una fase de análisis con la intención de entender el sistema y su estructura. Para definir esta estructura la dividimos en un conjunto de roles e interacciones entre ellos. Estos roles, a su vez, se componen de responsabilidades (definen la funcionalidad), permisos o recursos (lectura, escritura y/o generación de información) y protocolos (funciones que requieren de la interacción con otro rol). [7]

Este análisis de requisitos corresponde al que encontramos en el apartado previo.

Una vez modelados los roles y las interacciones del sistema podemos pasar a la fase de diseño. En este paso utilizamos los roles y las interacciones para definir tres modelos diferentes:

- Modelo de Agentes: Agrupamos roles relacionados para crear el agente que se encargará de ejecutarlos.
- Modelo de Comunicaciones: Se establecen las comunicaciones que habrá en el programa entre los diferentes agentes, sabiendo las interacciones entre los roles de cada uno.
- Modelo de servicios: Se especifica la relación entre los agentes. Diferenciando entre diferentes relaciones entre mismos agentes, más completo que el modelo de comunicaciones.

CAPITULO 4. Funcionamiento del programa

Una vez hemos explicado el problema que se quiere afrontar y también se han explicado las herramientas iniciales utilizadas para ello, es posible empezar con la descripción de lo que será el software implementado.

Se trata de un programa en el que se simulan los comportamientos y lo sucesos que ocurren en un aeropuerto con un A-CDM implementado.

El programa simula la evolución de los tiempos calculados de cada avión, desde que se definen hasta que llegan a suceder; estableciendo las relaciones necesarias entre los aviones y sus tiempos, para asegurar una correcta propagación de un retraso. De esta manera, se busca saber la evolución del tráfico aéreo y como se ve afectado frente a diferentes situaciones y cambios.

4.1 *Descripción general del software*

Para poder simular esta realidad, el programa se nutre de una serie de inputs, opera estos datos gracias a la inteligencia programada en cada uno de los agentes implementados, y da como resultado una serie de outputs, con lo que se pretende que el usuario pueda ver los resultados y analizarlos.
En la figura 4-1 se puede ver un esquema general de los archivos de entrada y de salida que nos ofrecerá el programa, una visión externa de lo que, a nivel de usuario, es el programa.

Figura 4-1: Visión general del Software

La serie de inputs necesarios en el programa se ha intentado que sea lo más básica posible, para dar posibilidad de poder repetir la simulación cambiando

parámetros fácilmente y para dificultar que el usuario se equivoque al introducirlos en el programa.

Estos documentos input se dividen en Tráfico Aéreo, obtenido directamente de la base de datos de Eurocontrol; Parámetros Aeropuerto, para dar la posibilidad de variar las simulaciones según el tipo de aeropuerto; y de Retrasos, que añadirán realismo a la simulación y darán al usuario la posibilidad de observar como el sistema responderá a una serie de situaciones adversas, además para este tipo de input se ha añadido una interfaz de usuario, lo que hace que sea prácticamente imposible introducir error al añadirlo. Otro input que se puede utilizar es un documento de resultados obtenido previamente en el programa, lo que dará la posibilidad de hacer una comparación entre diversas respuestas del sistema a diversas situaciones.

Después de la ejecución el programa nos guardará una serie de documentos con los Resultados de la simulación, un documento detallado con todo el tráfico aéreo y sus incidencias; un documento de Milestones de la simulación, para poder seguir el orden que ha seguido la simulación al ejecutarse tanto para aviones como para incidencias (retrasos) y un documento con los Parámetros de la simulación, con lo que guardar las condiciones con las que se realizó el estudio y poder repetir el experimento. Además, el programa ofrece la posibilidad de cargar unos resultados previos para poder hacer una comparación con más detalle entre simulaciones.

Esta comparación se puede hacer a través de la interfaz de análisis, a partir de la cual se pueden analizar cada uno de los parámetros (tiempos concretos de los aviones). Esta interfaz crea graficas comparativas y guarda cada uno de los Análisis de Parámetros en archivos para su posterior estudio.

4.2 *Estructura del programa*

El programa se ha estructurado como un conjunto de agentes e interfaces que trabajan de manera asíncrona para realizar correctamente las simulaciones.

Estas simulaciones se basan en el intercambio de información, en forma de mensajes entre agentes, para simular el comportamiento del tráfico aéreo en un aeropuerto.

Para que el programa realice correctamente las simulaciones se han implementado dos grandes grupos de agentes:

- Agentes CDM, que se encargan de simular el comportamiento real de los diferentes *stakeholders* del aeropuerto. Lo componen los agentes: CFMU, ATC, Airport, AO y GH.
- Agentes Simulación CDM, que se encargan de llevar la simulación. controlan el tiempo de la simulación y la evolución de la misma, y centralizan el intercambio de datos entre los agentes CDM. Lo componen los agentes InfoSharing y MilestoneTrigger.

- Agentes Software, se encargan de asegurar que la simulación se realiza correctamente. Dentro de este grupo encontramos 2 tipos:
    - Agentes Input: preparan las condiciones iniciales, manejan los datos de entrada y generan los archivos necesarios para realizar la simulación. Lo componen los agentes Simulator Input y DDR2.
    - Agentes Output: trabajan con los resultados de la simulación. Muestran la simulación al usuario, guardan los resultados en un fichero y permite al usuario analizarlos. Lo componen los agentes Simulador gráfica y Output.

Otra parte importante del programa son las interfaces. Estas interfaces nos permitirán actuar sobre los parámetros iniciales, ver la simulación en tiempo real y ver y analizar los resultados de la misma. Estas interfaces son:

- Interfaz Simulator Input, que nos permite administrar las condiciones iniciales del programa y añadir los archivos de entrada a la simulación.
- Interfaz Retrasos, para administrar los retrasos que se añadirán a la simulación.
- Interfaz simulación, que nos permitirá ver la evolución de la simulación en tiempo real.
- Interfaz Output, para ver los resultados de la simulación.
- Interfaz Análisis, nos permite analizar y comparar diferentes parámetros de la simulación.

En la figura 4-2 podemos ver un esquema general de los agentes y las interfaces que componen el programa.

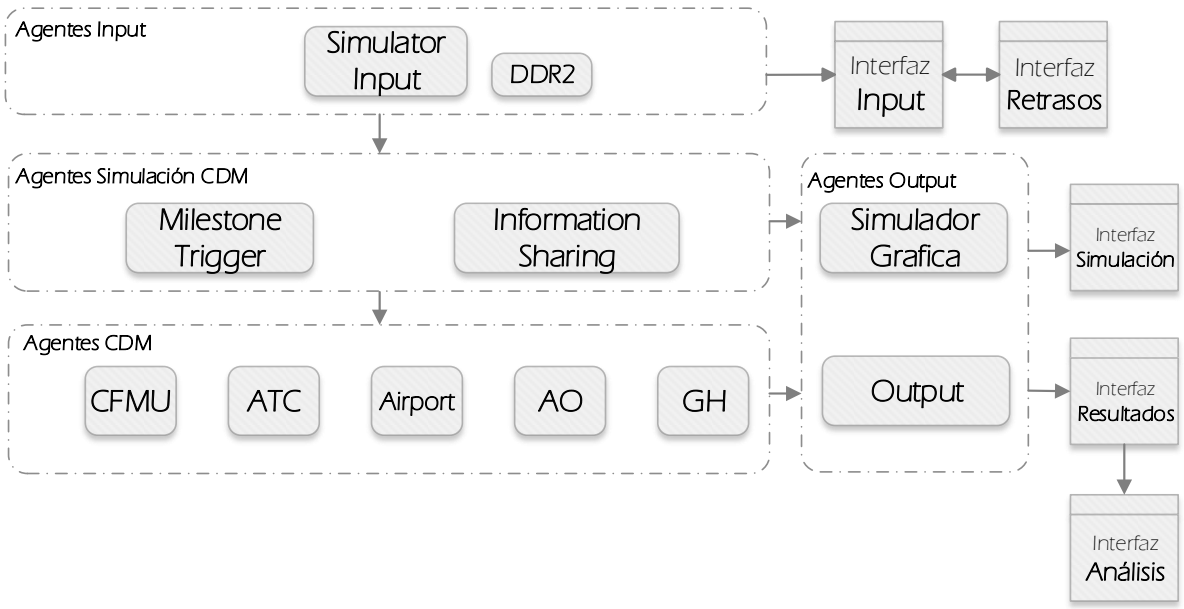


Figura 4-2: Esquema Estructura programa

Este esquema nos muestra los agentes y las interfaces del programa, y además representa las comunicaciones entre los diferentes grupos con flechas entre estos.

Comunicaciones

Al tratarse de un sistema multiagente, las comunicaciones entre agentes es una parte muy importante. Además, dado la naturaleza del A-CDM, es importante simular y monitorizar las comunicaciones; cosa relativamente sencilla en sistemas descentralizados y asíncronos como los que permite crear el middleware JADE.

En este caso en particular, podemos decir que en un CDM las comunicaciones se basan en intercambiar información de aviones, sus diferentes tiempos y actualizaciones. Por esta razón las comunicaciones entre agentes se han intentado simplificar a enviarse datos de aviones, o, como se ha implementado en el programa, se envían constantemente mensajes con *Aircraft*.

Este *Aircraft* es un objeto que contiene un avión y todos sus parámetros y variables; como su ID, sus tiempos de vuelo, la aerolínea a la que pertenece, etc. Que se codifica mediante Json y se envía de un agente a otro.

Debido a que en el programa no solo se intercambian datos de tráfico aéreo, sino que también hay muchas comunicaciones relacionadas con el funcionamiento del programa, para iniciar la simulación o establecer los parámetros iniciales en todos los agentes, por ejemplo; se ha optado por no implementar otro tipo de mensajes, sino dotar de más complejidad al objeto *Aircraft.* Añadiéndole un pequeño conjunto de variables que nos dan la posibilidad de utilizarlo para todos los tipos de comunicaciones. Simplemente añadiendo variables que nos indiquen que tipo de mensaje es y que nos permitan añadir la información adicional nos dan la posibilidad de simplificar las comunicaciones en gran manera.

El intercambio de estos objetos permite al programa una propagación ágil y segura de la información ya que se ha implementado en todos los agentes la capacidad de recibir, procesar, y enviar mensajes de manera asíncrona; es decir, cada agente reacciona a los mensajes que recibe.

El middleware JADE nos permite trabajar con los mensajes de manera sencilla, simplificando la creación y envío de mensajes a definir parámetros como los receptores, el tipo de mensaje y añadir el contenido.

El contenido del mensaje se ha simplificado para enviar únicamente un Json que contiene el objeto avión con todos sus variables y valores. De esta manera, en el envío y recepción de mensajes encontramos un proceso de codificación/descodificación de un objeto *Aircraft*.

Para la lista de receptores de cada mensaje y el tipo de mensaje se ha seguido las reglas FIPA de mensajes en sistemas multiagente, explicado más en profundidad en el apartado de comunicaciones.

Para almacenar, administrar y operar con toda esta información de aviones, los agentes cuentan con tablas internas donde guardarlos. De manera que todos los agentes (que lo requieren) tienen una tabla interna con los datos del tráfico aéreo actualizados en todo momento, y cuentan con la información necesaria para realizar sus comportamientos correctamente.

Agentes

Esta forma de funcionar de manera asíncrona, reaccionando a estímulos (mensajes) del entorno, afecta directamente en la estructura de los agentes. Se puede definir las actuaciones de los agentes, de manera simplificada, como en la figura 4-3:

Figura 4-3: Estructura interna agente

En esta figura se puede apreciar como los agentes solo iniciarán a procesar datos al recibir un mensaje. En ese momento lo decodifican para crear un objeto avión, que analizan para saber si se trata de un mensaje con información de alguno de los aviones del tráfico aéreo o de información del programa pero ajena al CDM.

Una vez procesada esta información el agente procede a administrar los datos internos (la tabla de aviones o los parámetros externos al CDM, según el mensaje que se haya recibido) y a preparar y enviar el mensaje de respuesta al agente que toque.

La parte más importante del agente es el “Procesamiento información / Ejecución comportamientos del agente”, se puede definir como la inteligencia del agente, es donde el agente lleva a cabo las funciones y roles que tiene asignados en el programa.

La manera de diferenciar cada uno de los agentes es que todos tienen una inteligencia distinta, hacen cosas diferentes. De esta manera se ha hecho la clasificación en los 4 grupos de agentes: agentes CDM, agentes Simulación CDM, y los agentes software (input y output).

Agentes CDM

Los encargados de simular la evolución del tráfico aéreo a través de un aeropuerto con CDM implementado son los agentes CDM. Estos agentes comparten una gran parte de los roles y los atributos, pues todos ejecutan la misma serie de funciones.

Como podemos ver en el esquema de la figura 4-4, la diferencia entre los comportamientos de estos agentes CDM se basa en las comprobaciones y las actuaciones que hacen sobre los tiempos del avión. Esto se debe a que cada uno de los agentes representa un *stakeholder* del aeropuerto distinto (CFMU, ATC, Aeropuerto, Aircraft Operator y Ground Handling).

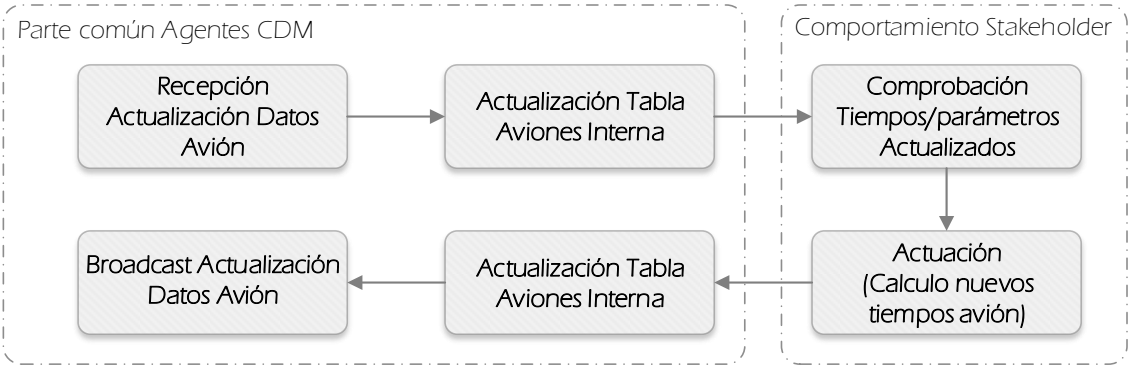


Figura 4-4: Esquema comportamiento Agente CDM

Estos comportamientos se basan en comprobar que tiempos del avión se han actualizado y actuar en consecuencia, calculando otros tiempos o definiendo/actualizando el tiempo de algún milestone del avión. Cada uno de los agentes buscara actualizaciones de tiempos diferentes de un avión y actualizara diferentes tiempos/milestones. Por ejemplo: cuando reciba un avión con el tiempo de aterrizaje actualizado, el agente *Airport* hará los cálculos necesarios para obtener el tiempo de llegada a *gate*, y en ese momento mandará un mensaje avisando de que ha actualizado ese tiempo.

De esta manera conseguimos que todos los agentes escuchen constantemente las actualizaciones de tiempos de todos los aviones y actúen en consecuencia, calculando los tiempos de los cuales están a cargo. Esto hace que durante la simulación haya un constante tráfico de mensajes y en todo momento se estén actualizando los datos del tráfico aéreo.

Los 5 agentes CDM se encargan de tiempos y de milestones diferentes. Se ha hecho un trabajo previo para definir las responsabilidades de cada uno de los agentes, para así conseguir un comportamiento lo más real posible.

Esta asignación de tiempos, milestones y diferentes responsabilidades se puede encontrar explicada en el apartado de agentes, más adelante en este documento. En el Apéndice A, se puede ver que también se ha hecho un trabajo

de análisis de que tiempos se definen/actualizan en cada milestone, así como que actuaciones realiza cada agente.

Agentes Simulación CDM

Para realizar esta simulación correctamente se necesitan dos funciones fundamentales: Un agente que se encargue de centralizar el flujo de datos, que se encargue de recibir las actualizaciones y hacer un *broadcast* a todos los demás agentes; y un agente que se encargue de ejecutar la simulación, que tenga un reloj y empiece, cronológicamente, los milestones de todos los aviones, para que la simulación avance.

Otra peculiaridad de estos dos agentes es el hecho de que ejecutan dos comportamientos en paralelo, mientras que todos los demás agentes centran toda su ejecución en solo un comportamiento. Este hecho se debe a que estos dos agentes en particular tienen que manejar un gran flujo de datos, superior al que manejan los demás agentes. Para evitar que estos agentes se colapsen se han implementado estos dos comportamientos en paralelo, uno para administrar las comunicaciones y otro para ejecutar las funciones definidas del agente.

El agente InfoSharing se ha definido a partir de la guía de implementación de Eurocontrol [4]. Este agente se encarga de recibir todas las actualizaciones de datos de aviones, registrarlas, y enviar la información a todos los agentes CDM. De esta manera conseguimos que en todo momento todos los agentes tengan los mismos datos, actualizados constantemente, en su tabla de aviones interna. Además, este agente se encarga de enviar estas actualizaciones de tiempos a los agentes output Simulador Gráfica y Output, para que muestren los resultados en una interfaz gráfica y guarden los resultados finales en un documento.

Dado que está constantemente recibiendo mensajes de actualización, este agente necesita tener un comportamiento que se encarga de recibir, procesar y guardar los datos recibidos en los mensajes (Comportamiento “Comunicaciones”); y otro comportamiento que va cogiendo los datos guardados, los procesa para actualizar su tabla interna y envía las actualizaciones a todos los agentes (Comportamiento “AdministrarTabla”).

En la figura 4-5 podemos ver el comportamiento de este agente:

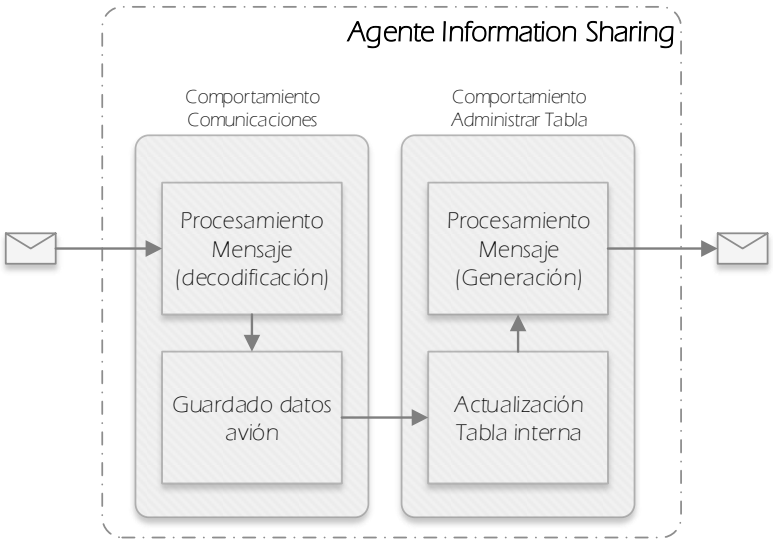


Figura 4-5: Esquema Ejecución Information Sharing

El otro agente Simulación CDM, Milestone Trigger, se encarga de llevar la ejecución de la simulación. Este agente tiene la información de cuando se llevarán a cabo cada uno de los milestones de los aviones y se encarga de iniciar (ejecutar) cada uno de estos milestones por orden cronológico. También se encarga de generar el archivo "MilestonesResultado.txt", en el cual podremos ver los milestones de la simulación en el orden en el que se han ejecutado.

De esta manera tenemos un agente que se encarga de llevar el tiempo de la simulación y con el que podemos definir el tiempo que durará la simulación, definiendo más o menos velocidad al reloj que decide cada cuanto se irá ejecutando cada milestone de la lista.
Al llegar al final de la lista, cuando se han acabado los milestones, Milestone Trigger prepara y envía un mensaje a los agentes del programa avisando de que se ha acabado la simulación.

Estos milestones se definen (y actualizan) a partir de diversos tiempos de un avión, y por tanto, los calculan los agentes CDM. Esto hace que Milestone Trigger reciba constantemente información sobre actualizaciones de estos milestones y, igual que Information Sharing, tenga que manejar un flujo de comunicaciones elevado. Además, tiene que hacer un procesamiento extra que Information Sharing no tenía que hacer, tiene que reordenar cronológicamente toda su lista de milestones cada vez que le llega una actualización de algún milestone.

Por esto se han implementado dos comportamientos en paralelo en este agente. El primero se encarga de recibir y decodificar los mensajes, para luego reordenar la tabla de milestones entera (Comportamiento "Administrar Milestones"); y el segundo comportamiento se encarga de analizar cada uno de los milestones (a cada tic del reloj) para preparar y enviar el mensaje que hará que ese milestone se ejecute en el programa (Comportamiento "TriggerMilestones"). La diferencia entre ellos es que mientras el primero solo se ejecuta al llegar un mensaje; el segundo se está ejecutando constantemente, procesando los mensajes que el

otro comportamiento ha descodificado y guardado en la lista interna, hasta que se acaban.

En la figura 4-6 podemos ver el comportamiento de este agente:

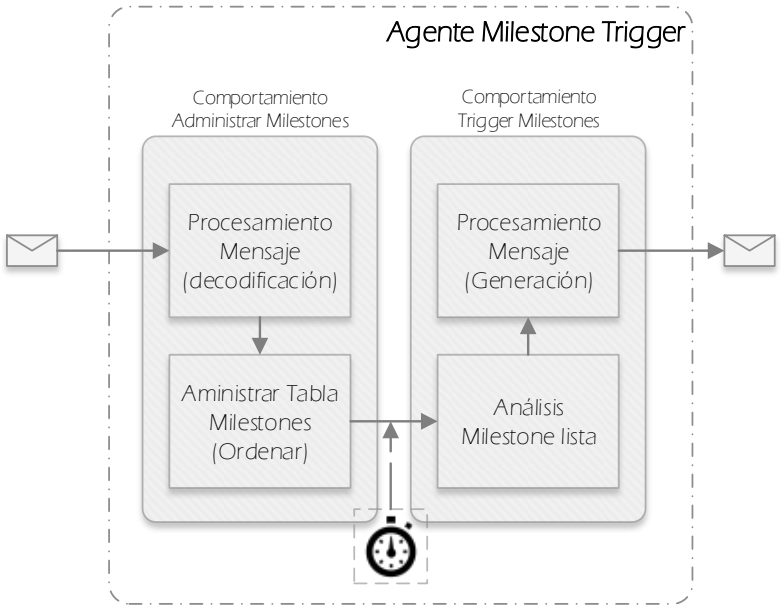


Figura 4-6: Esquema Ejecución Milestone Trigger

Agentes Software

Los agentes antes comentados son los encargados de ejecutar la simulación del A-CDM, pero en el programa también tenemos un grupo de agentes que se encargan de preparar la simulación y guardar y mostrar los resultados de esta, los llamados agentes software. Estos agentes también se diferencian de los anteriores en que además de manejar comunicaciones y ejecutar sus funciones, también tienen conexiones con diferentes interfaces gráfica; con las que reciben parámetros, muestran resultados y dan diferentes opciones al usuario.

El primer grupo de agentes software lo componen los agentes que se encargan de preparar la simulación. Leen los documentos de entrada y reciben los parámetros iniciales del usuario para después generar los archivos necesarios para la simulación y avisar a los agentes que simulan de las condiciones iniciales.

El agente más importante en este grupo es el llamado Simulator Input. A partir de su “Interfaz Simulator Input” recibe los diferentes parámetros iniciales que se necesitarán para la simulación, el más importante de estos es la ruta donde se encuentran los archivos input del programa. Una vez con estos parámetros, procede a cargar, leer y analizar los archivos input para, a partir de ellos, generar los archivos input que utilizará el programa. Este agente, a partir de los archivos input (Parámetros Aeropuerto y Tráfico Aéreo) genera los siguientes archivos:

- Flight Plans / Flight Plans Salidas: guarda la información que encontraríamos en los Flight Plans de estos vuelos.

- AO1/AO2/Vuelos Aerolínea: Organiza los vuelos por aerolíneas y junta los Flight Plans de entrada con los de salida. De esta manera se obtiene un archivo con los datos de aterrizaje y despegue de cada avión.
- TriggerCDM: Define los milestones cuyos tiempos se pueden calcular a partir de los Flight Plans de cada avión (Milestones 1-6).
- Retrasos: Lista de los retrasos que se ejecutaran en la simulación.

Además de generar los archivos que se utilizarán para la simulación, Simulator Input también recoge los parámetros iniciales, los envía a todos los agentes y los escribe en uno de los documentos output (ParametrosSimulacion.txt).

Otra función de este agente es la de administrar los retrasos que se añadirán a la simulación. A través de la “interfaz retrasos” permite al usuario crear, borrar, modificar y cargar desde archivo diferentes retrasos para que luego se añadan y se ejecuten en la simulación.

Una vez el agente ha creado todos los archivos necesarios y el usuario ha seleccionado “Iniciar simulación”, Simulator Input envía un mensaje avisando a todos los agentes de que la simulación empieza.

Para conseguir el documento input de tráfico aéreo que se utilizará para generar los archivos input anteriores tiene que haber un trabajo previo sobre el documento de entrada de tráfico aéreo obtenido de la base de datos de Eurocontrol (DDR2). Este trabajo lo realiza el agente DDR2, que recibe del Simulator Input la ruta del archivo y se encarga de leerlo, cargar, analizarlo y generar el archivo INPUT.txt que Simulator Input utilizará.

Para entender un poco mejor este proceso previo a la simulación, en la figura 4-7 encontramos un diagrama con las actuaciones de Simulator Input y DDR2. En esta figura se ve el proceso de inicio del programa hasta el momento de iniciar la simulación, los eventos están ordenador temporalmente en orden descendente.

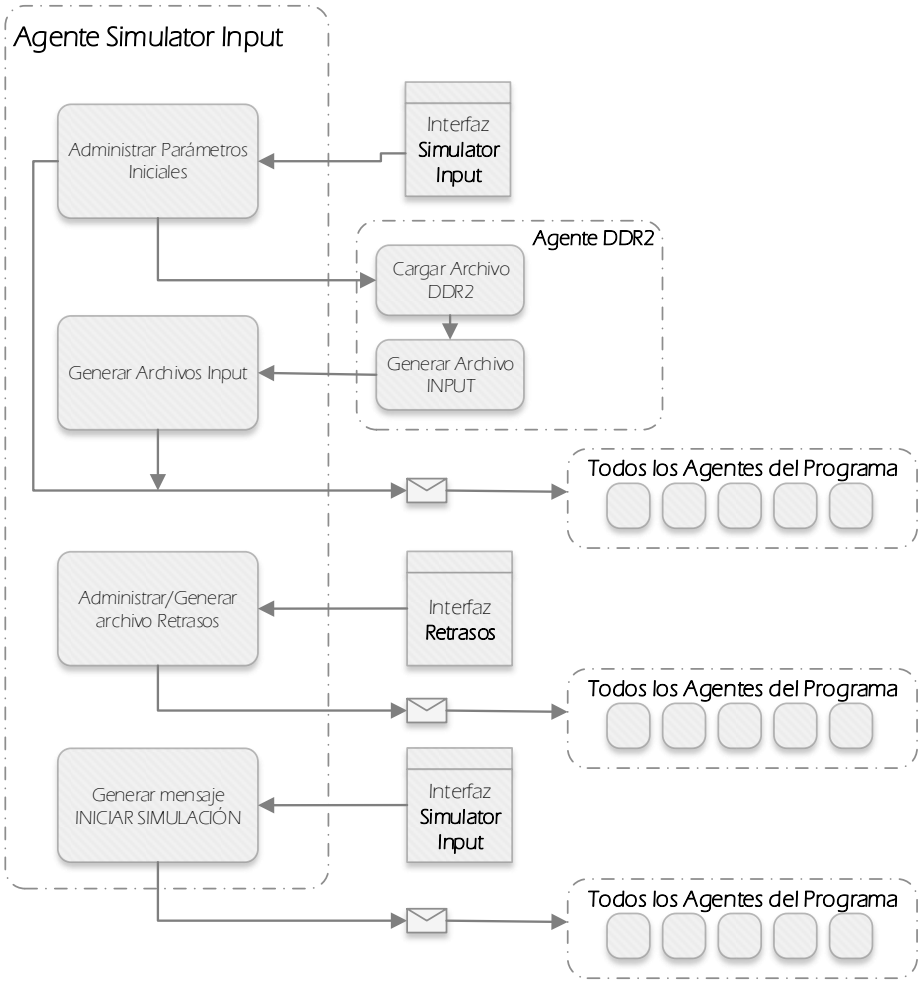


Figura 4-7: Esquema Temporal Ejecución Agente Simulator Input

De la misma manera que hay agentes que se encargan de que la ejecución del programa empiece correctamente, encontramos un grupo de agentes que se encargan de asegurar de que la ejecución se termina correctamente, se guardan los resultados y se muestran al usuario.

El primero de estos agentes software output es el agente Simulador Gráfica. Este agente recibe constantemente las actualizaciones de datos desde el Information Sharing y muestra la evolución de la simulación al usuario a través de su interfaz gráfica. Además de administrar y mostrar la tabla de aviones, este agente está preparado para parar la ejecución en el momento que el usuario lo pida. Al clicar el botón “Terminar Simulación” el agente prepara y envía un mensaje para avisar de que se termina la simulación, y el programa reacciona parando la simulación y generando todos los archivos de resultados.

Al terminar la simulación, el agente encargado de generar el archivo que guarda los resultados de la simulación es el Agente Output. Además de generar el archivo results.txt, este agente utiliza su interfaz gráfica para mostrar al usuario una tabla con los resultados obtenidos en la simulación.

En la interfaz de resultados, aparte de observar los resultados de la simulación, se permite al usuario cargar otro documento de resultados. Cargando este documento se da la posibilidad al usuario de comparar datos de diferentes simulaciones y observar los cambios que se han producido.

A partir de esta interfaz output se puede abrir la Interfaz Análisis. A través de la cual podemos realizar un análisis temporal del número de aviones que se han retrasado, seleccionando el parámetro/tiempo que queremos observar. Además, también podemos añadir filtros al análisis, para observar solo los aviones que nos interesan (filtro por modelo y aerolínea) o solo la ventana de tiempo que nos interesa.

Como podemos ver en la figura 4-8, la interfaz análisis nos permite definir estos filtros y elegir el parámetro que queremos analizar. Nos muestra un plot del número de aviones afectados (que están sufriendo un retraso) por minuto.

También nos calcula diferentes parámetros del análisis como el número total de minutos retrasados, la media de minutos de retraso por avión o el avión que ha sufrido un mayor retraso.

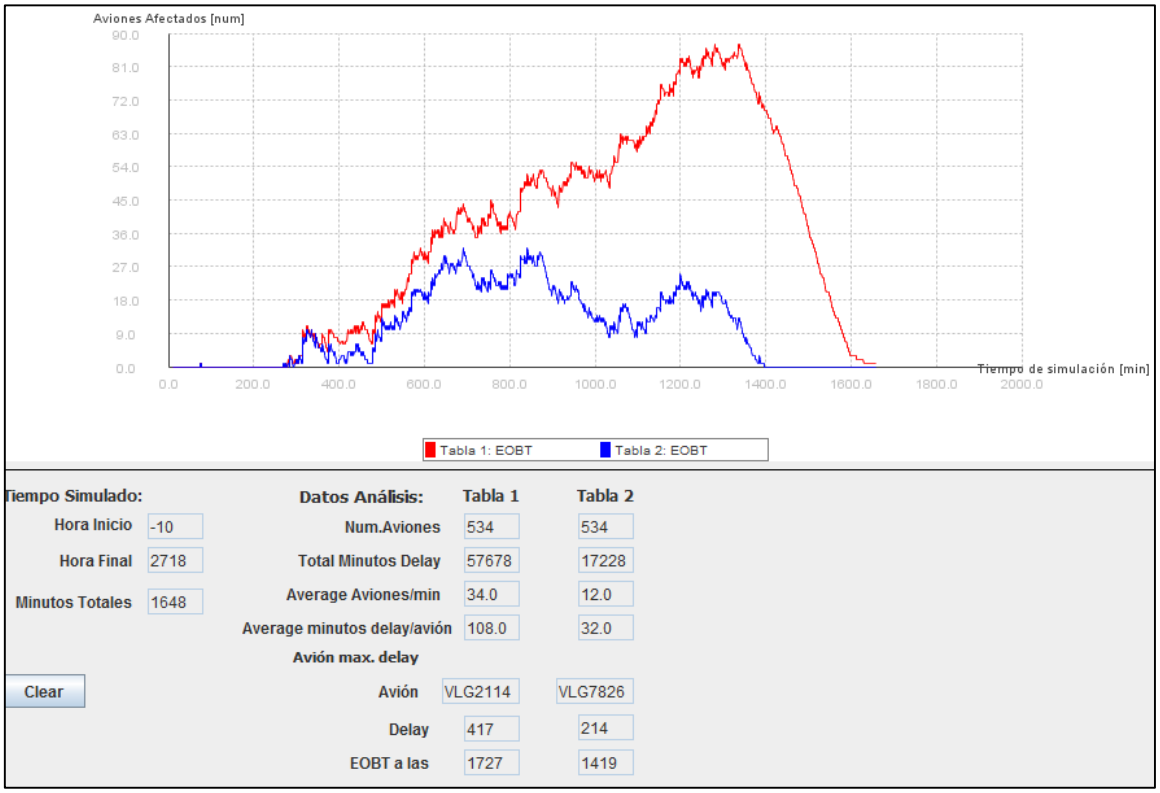


Figura 4-8: Fragmento Interfaz Análisis

4.3 *Agentes*

Como ya se ha explicado anteriormente, dada la naturaleza del sistema a simular y del middleware utilizado, los agentes se ejecutan de manera asíncrona y distribuida. Esto quiere decir que estos agentes no se ejecutan línea a línea, sino que reaccionan a estímulos (mensajes).

Esto afecta directamente en su estructura, que la podríamos dividir en 3 partes: Recepción mensajes, ejecución comportamientos y envío mensajes.

Dado que las dos partes de comunicaciones prácticamente iguales en todos los agentes y se explicarán en el apartado de Comunicaciones de la memoria, en este apartado nos centramos en explicar el "Procesamiento información / Ejecución comportamientos del agente", que se puede definir como la inteligencia del agente, es donde el agente lleva a cabo las funciones y roles que tiene asignados en el programa. Ver figura 4-9:

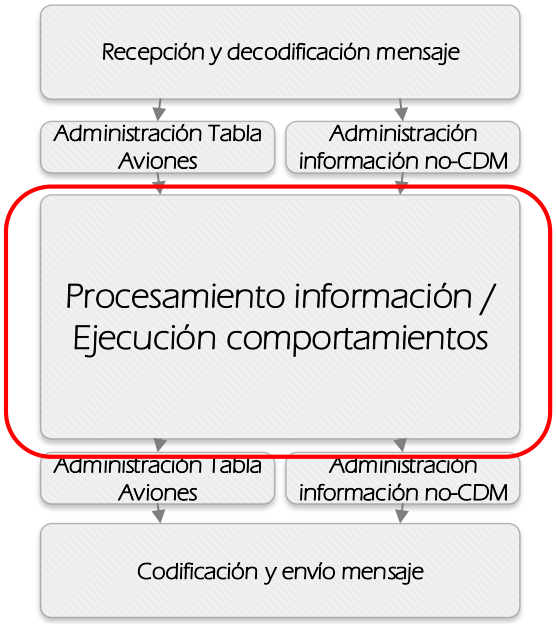


Figura 4-9: Esquema ejecución Agente

Esta ejecución se realiza de forma iterativa, ya que al llegar un mensaje el agente lo procesa, procesa la información y ejecuta comportamientos y después envía la respuesta, y por último, vuelve al principio, donde espera hasta que le vuelve a llegar otro mensaje.

La manera de diferenciar cada uno de los agentes es que todos tienen una inteligencia distinta, hacen cosas diferentes. De esta manera se ha hecho la clasificación en los 4 grupos de agentes: agentes CDM, agentes Simulación CDM, y los agentes software (input y output).

Para cada uno de estos agentes se basará en explicar los aspectos que los hacen diferentes al resto:

- Comportamientos que ejecuta el agente. Así como las diferentes funciones que utiliza y una explicación general de como realiza sus cálculos.
- Archivos que carga, lee, escribe o genera al ejecutar sus comportamientos.
- Constantes que utiliza para su ejecución, tanto si son recibidas a través de los parámetros iniciales como si están definidas directamente en el código.
- Parámetros o variables más importantes que utiliza.
- Interfaces que abre y administra el agente, así como una explicación de que hacen y como.

4.3.1 Agentes CDM

Este primer grupo se corresponde a los agentes encargados de realizar la simulación del A-CDM en sí. Estos 5 agentes se corresponden a los *stakeholders* definidos por Eurocontrol en su manual [4] y de los que ya hemos hablado en el capítulo 2.

Estos agentes se encargan de procesar las diferentes actualizaciones de los aviones para hacer los diferentes cálculos de tiempos de los cuales están a cargo. De esta manera conseguimos que cada uno de los agentes CDM analice y actualice los datos del avión de los cuales se hace cargo en la realidad el *stakeholder* al cual representa. Estos análisis de tiempos se pueden producir en dos circunstancias: definición de los tiempos de un avión o propagación de un retraso en alguno de los tiempos de un avión.

De esta manera obtenemos un conjunto de agentes que es capaz de definir todos los tiempos de un avión únicamente a partir de los datos de entrada al simulador (los Flight Plans de los vuelos); y que también es capaz de detectar y analizar un retraso en cualquiera de los tiempos de un avión y realizar una propagación de este retraso en todos los demás tiempos de manera correcta y lo más real posible. Como podremos ver más adelante, las comprobaciones que se realizan para definir tiempos y para propagar retrasos no tienen por qué coincidir, de hecho los agentes tienen un número mayor de comprobaciones para propagar un retraso que para definir tiempos.

En relación a este cálculo y actualización de tiempos del avión encontramos los milestones. Como hemos visto antes, estos milestones ocurren en tiempos concretos del ciclo CDM de un avión y, por tanto, se deben actualizar a la vez que actualizamos los tiempos de dicho avión. Por tanto, cada agente tiene implementados unos comportamientos que relacionan los tiempos con los milestones y que hacen que actualice ambas cosas a la vez.

Además de actualizarlos, los milestones se utilizan como *trigger* para la simulación de cada avión. Cada vez que se llega al momento de un milestone de un avión, este es ejecutado por agente *MilestoneTrigger*. Estas ejecuciones de milestones se realizan en forma de avisar a alguno de los agentes CDM de que ese avión ha llegado a dicho milestone. Para la decisión de que agente ejecuta cada uno de los milestones se ha realizado un estudio para conseguir un mayor realismo y que cada agente reciba y ejecute los milestones que le tocan (por

ejemplo: el agente ATC recibirá el mensaje de ejecutar el milestone 4 (TFIR), ya que en la realidad es ATC quien detecta que el avión ha entrado en la FIR del aeropuerto).

Debido a que los roles de estos agentes son muy parecidos, así como su estructura y sus comportamientos, se ha optado por la creación de un *agente CDM* del cual heredan. Este agente engloba todos los comportamientos que son iguales en estos agentes CFMU, ATC, Airport, AO y GH.

En la figura 4-10 podemos ver como estos agentes se parecen en el procesamiento de los mensajes recibidos y enviados, y en la manera de guardar los datos. Las diferencias se basan en que tiempos se actualizan y de qué manera lo hace cada uno de los agentes.

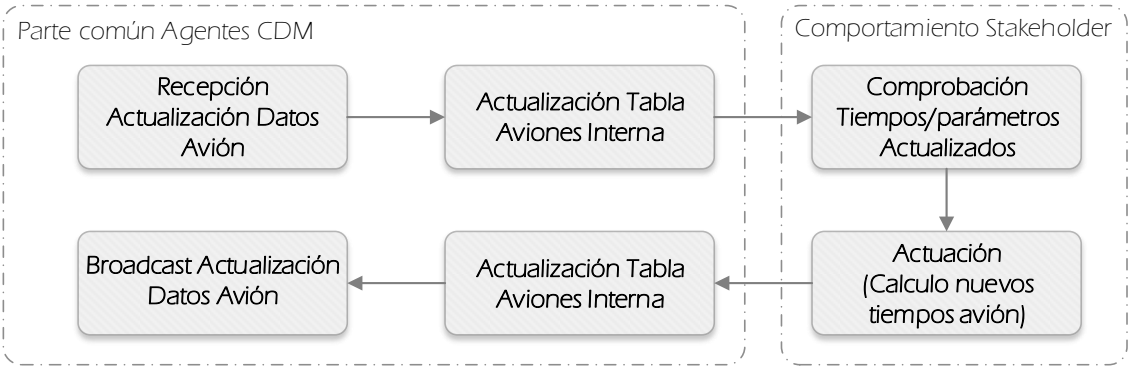


Figura 4-10: Esquema ejecución Agentes CDM

Para explicar correctamente lo que hace cada uno de estos agentes CDM, en los siguientes apartados se ha seguido una estructura diferente a la de explicar los demás tipos de agentes.

La estructura para explicar estos agentes CDM será:

- Descripción breve de en qué consiste el agente y de que se encarga en el sistema.
- Archivos que utilizará para cargar datos o para hacer los cálculos.
- Milestones que ejecuta. Enumerando que milestones es el encargado de ejecutar y recibe del agente *MilestoneTrigger*.
- Tiempos definidos. Listando los tiempos del avión que el agente se encarga de definir, además de una breve explicación de cómo los calcula.
- Milestones que actualiza/define. De la misma manera que los tiempos, los milestones que el agente calcula a partir de los tiempos del avión.
- Actualización de tiempos. Tiempos que analiza y actualiza para propagar un retraso.
- Constantes que utiliza para la ejecución, tanto si son recibidas a través de los parámetros iniciales como si están definidas directamente en el código.
- Parámetros o variables importantes que utiliza.
- Comportamientos. Explicación de otros comportamientos que realice el agente aparte de analizar y actualizar tiempos.

4.3.1.1 CFMU

- **Descripción**: Este agente simula los comportamientos del CFMU, el agente que supervisa el espacio aéreo y los planes de vuelo (Flight Plans) de los aviones de la simulación. Por esto podremos ver que este agente actúa en los primeros milestones de un avión, definiendo los tiempos a partir del plan de vuelo y supervisando el avión mientras vuela.
- **Archivos**: El agente CFMU leerá 2 documentos al iniciar la simulación. El FlightPlans.txt y el FlightPlansSalidas.txt. Al leer estos 2 documentos el agente obtendrá toda la información necesaria para crear el objeto Aircraft de cada avión al recibir que un avión acaba de realizar el primer milestone y aparece por primera vez en la simulación.
- **Ejecuta los milestones**:
  - Milestone 1 → 3 horas antes del ETOT'
  - Milestone 2 → 2 horas antes del ETOT'
- **Tiempos definidos**: El CFMU define por primera vez los tiempos de:
  - ETOT' → Con la información obtenida en el Flight Plan
  - ELDT → Con la información obtenida en el Flight Plan
  - TFIR → Con la información obtenida en el Flight Plan
  - TAPP → Con la información obtenida en el Flight Plan
- **Define/Actualiza los milestones:**
  - Milestone 4 (TFIR) → cuando se ha actualizado ETOT'
  - Milestone 5 (TAPP) → cuando se ha actualizado TFIR
  - Milestone 6 (ELDT) → cuando se ha actualizado TAPP
- **Actualizaciones de tiempos**:
  - TFIR → cuando se ha actualizado ETOT'
  - TAPP → cuando se ha actualizado TFIR
  - ELDT → cuando se ha actualizado TAPP
- **Constantes**:
  - rutaTXT: ruta para encontrar el documento FlightPlans.txt
  - rutaTXTS: ruta para encontrar el documento FlightPlansSalidas.txt
  - FlightPlans: Contiene la información de los flight plans de los aviones que llegan al aeropuerto
  - FlightPlansSalidas: Contiene la información de los flight plans de los aviones que salen del aeropuerto
  - avionesSimulacion: entero que indica la cantidad de aviones que se simularán
  - ModelosCat y NumCat: contienen la información de la tabla de categorías de cada avión según el modelo de este.

- **Parámetros**:
    - Tabla: Contiene la lista de los aviones que se están simulando con los parámetros de cada uno de ellos, se actualiza constantemente según avanza la simulación.
    - Retrasoavion: Lista de retrasos tipo 2 y tipo 3 que aplicará el agente durante la simulación
    - Retrasotiempo: Lista de retrasos tipo 1 que aplicará el agente durante la simulación
- **Comportamientos**:
    - Este agente es el encargado de definir un avión en el simulador por primera vez cuando este está realizando su primer milestone. Por este motivo este agente tendrá de realizar una serie de acciones que los otros agentes no realizarán. Este agente será el que escribirá todos los parámetros del nuevo avión, desde su ID, su modelo, aerolínea, aeropuertos donde actuará… (información que se encuentra en el Flight Plan del avión).
    - También será el encargado de buscar según el modelo de cada avión y la tabla de categorías (ModelosCat y NumCat) la categoría del avión de estudio.
    - Cuando un avión solo sale de nuestro aeropuerto este agente también será el encargado de definir el objeto Aircraft, pero en este caso y de manera excepcional será el encargado de definir el tiempo ETOT del avión, pues esta información la sacará del documento FlightPlansSalidas.txt

4.3.1.2 **ATC**

- **Descripción:** Agente que simula el Air Traffic Controller (ATC), administra los aviones del tráfico aéreo en vuelo y en pistas, controlando cuando despegan y aterrizan. Desde este agente se ejecutarán todos los milestones que suceden en el aire, considerando aire desde el momento en que el avión enciende motores hasta que llega al Gate del aeropuerto de destino.

  Este agente también es el encargado de controlar la pre-departure sequence, administrando el flujo de aterrizajes y despegues para asegurar que el número de operaciones hora no sobrepasa la capacidad de la pista.
- **Archivos**: El agente ATC no leerá ningún documento
- **Ejecuta los milestones**:
    - Milestone 3 → ETOT'
    - Milestone 4 → TFIR

  - Milestone 5 → TAPP
  - Milestone 6 → ELDT
  - Milestone 7 → EIBT
  - Milestone 9 → Confirmación EOBT
  - Milestone 10 → Definición TSAT
  - Milestone 14 → TSAT
  - Milestone 15 → EOBT
  - Milestone 16 → ETOT
- **Tiempos definidos**: El ATC define por primera vez el tiempo de:
  - TSAT → Lo define cuando conoce el tiempo de EOBT
- **Define/Actualiza los milestones:**
  - Milestone 9 → cuando se ha actualizado EIBT
  - Milestone 10 → cuando se ha actualizado EIBT
  - Milestone 14 (TSAT) → cuando se ha actualizado EOBT
  - Milestone 16(ETOT) → cuando se ha actualizado ETOT
  - Milestone 6(ELDT) → cuando se ha actualizado ELDT
- **Actualizaciones de tiempos**:
  - TSAT → cuando se ha actualizado EOBT
  - ETOT → cuando se ha actualizado ETOT
  - ELDT → cuando se ha actualizado ELDT
- **Constantes**:
  - avionesSimulacion: entero que indica la cantidad de aviones que se simularán
  - politicaCongestio: entero que indica la política que se seguirá al definir la pre-departure Sequence (1 = el avión es retrasado hasta encontrar un slot vacío, 2 = el avión se coloca en su slot y todos los aviones se retrasan manteniendo el orden de llegadas previsto)
  - TakeOffSequence: Secuencia de salidas del aeropuerto
  - LandingSequence: Secuencia de llegadas al aeropuerto
  - SeparationTimein: Separación mínima en minutos de los aviones que llegan según su categoría.
  - SeparationTimeout: Separación mínima en minutos de los aviones que salen según su categoría.
  - tiempoMil9: Minutos entre EIBT y el milestone 9
  - tiempoMil10: Minutos entre el milestone 9 y el milestone 10
  - EOBTtoTSAT: Minutos necesarios entre TSAT y EOBT por encendido de motores.

- **Parámetros**:
    - Tabla: Contiene la lista de los aviones que se están simulando con los parámetros de cada uno de ellos, se actualiza constantemente según avanza la simulación.
    - Retrasoavion: Lista de retrasos tipo 2 y tipo 3 que aplicará el agente durante la simulación
    - Retrasotiempo: Lista de retrasos tipo 1 que aplicará el agente durante la simulación
    - Retrasopista: Lista de retrasos tipo 4 que aplicará el agente durante la simulación.
    - SeparationsTimeReducted: Lista con las Separaciones mínimas entre aviones cuando estas sufren modificaciones durante un período de tiempo
- **Comportamientos**:
    - Este agente administra la tabla con las pre-departure sequence en todo momento. Cada vez que un avión quiere realizar una acción en la pista, ya sea llegada o salida, será el encargado de comprobar que a esa hora la pista estará libre y se puede usar. En caso contrario tiene 2 opciones según la politicaCongestio:
      -politicaConestio=1: realizará un escaneado a toda la información sobre todos los aviones que quieren llegar o salir del aeropuerto para encontrar un slot en donde posicionar ese avión.
      -politicaCongestio=2: El avión se coloca en el slot deseado y se retrasan todos los aviones que quieren actuar después para asegurar los márgenes de tiempo entre aviones.

4.3.1.3 **Airport**

- **Descripción:** Simula los comportamientos de las autoridades del aeropuerto. Se encarga de administrar los aviones cuando en movimiento en tierra, calculando tiempos de taxi de llegada y de salida.
- **Archivos**: El agente Airport no leerá ningún documento
- **Ejecuta los milestones**: El agente Airport no ejecuta ningún milestone
- **Tiempos definidos**: El Airport define por primera vez el tiempo de:
    - EIBT → Lo define cuando conoce el tiempo de ELDT
    - EOBT → Lo define cuando conoce el tiempo de ETOT
- **Define/Actualiza los milestones:**
    - Milestone 7(EIBT) → cuando se ha actualizado ELDT
    - Milestone 15(EOBT) → cuando se ha actualizado ETOT
    - Milestone 16(ETOT) → cuando se ha actualizado ETOT o EOBT

- **Actualizaciones de tiempos**:
  - EIBT → cuando se ha actualizado ELDT
  - EOBT → cuando se ha actualizado ETOT
  - ETOT → cuando se ha actualizado ETOT o EOBT
- **Constantes**:
  - avionesSimulacion: entero que indica la cantidad de aviones que se simularán
  - TSATtoEOBT: Minutos necesarios entre TSAT y EOBT por encendido de motores.
  - TaxiInTime: minutos necesarios para realizar el Taxi para llegar a la Gate
  - TaxiOutTime: minutos necesarios para realizar el Taxi desde la Gate hasta la pista de despegue.
- **Parámetros**:
  - Tabla: Contiene la lista de los aviones que se están simulando con los parámetros de cada uno de ellos, se actualiza constantemente según avanza la simulación.
  - Retrasoavion: Lista de retrasos tipo 2 y tipo 3 que aplicará el agente durante la simulación
  - Retrasotiempo: Lista de retrasos tipo 1 que aplicará el agente durante la simulación
- **Comportamientos**:
  - Asegurar que el tiempo de Taxi previsto para los aviones es plausible. Este agente tiene información sobre el tiempo mínimo para realizar los trayectos de Taxi, tanto de entrada como de salida del aeropuerto (entrados por el usuario mediante Interfaz). Esta agente se asegurará de que ningún avión tenga previsto llegar a Gate en menos tiempo del mínimo necesario.

4.3.1.4 **Aircraft Operator (AO)**

- **Descripción:** Simula los comportamiento de las Aerolíneas y sus decisiones sobre sus aviones.

  En el programa encontraremos más de un agente AO, ya que cada uno de ellos representa una aerolínea. Además encontraremos un agente AO-GEN, que se encargará de administrar los aviones que pertenezcan a una aerolínea sin agente, de esta manera nos aseguramos que no haga falta añadir un agente por cada aerolínea del aeropuerto, solo añadiremos las aerolíneas que más aviones tengan o las que más no interesen.

- **Archivos**: El agente AO leerá el documento VuelosAerolinea.txt que contiene información sobre las horas de las llegas y de las salidas de los aviones.
- **Ejecuta los milestones**:
  - Milestone 11→ ASBT
  - Milestone 12→ ARDT
  - Milestone 13→ ASRT
- **Tiempos definidos**: El AO define por primera vez el tiempo de:
  - ARDT→ Lo define cuando conoce el tiempo de ASBT o TSAT
  - ASRT→ Lo define cuando conoce el tiempo de ARDT
  - ETOT → Lo define cuando se define el avión con la información del documento VuelosAerolinea.txt
- **Define/Actualiza los milestones:**
  - Milestone 11(ASBT) → cuando se ha actualizado EOBT
  - Milestone 12(ARDT) → cuando se ha actualizado ASBT o TSAT
  - Milestone 13(ASRT) → cuando se ha actualizado ARDT
  - Milestone 14(TSAT) → cuando se ha actualizado ASRT
  - Milestone 16(ETOT) → Lo define cuando se define el objeto Aircraft.
- **Actualizaciones de tiempos**:
  - ASBT → cuando se ha actualizado EOBT
  - ARDT → cuando se ha actualizado ASBT o TSAT
  - ASRT → cuando se ha actualizado ARDT
  - TSAT → cuando se ha actualizado ASRT
- **Constantes**:
  - rutaVuelosAerolinea: ruta para abrir el documento de VuelosAerolinea.txt
  - avionesSimulacion: entero que indica la cantidad de aviones que se simularán
  - EOBTtoASBT: Minutos necesarios entre ASBT y EOBT
  - TSATtoARDT: Minutos necesarios entre ARDT y TSAT
  - ARDTtoASRT: Minutos necesarios entre TSAT y EOBT
  - ASBTtoARDT: Minutos necesarios entre TSAT y EOBT
  - ASRTtoTSAT: Minutos necesarios entre ASRT y TSAT
- **Parámetros**:
  - Tabla: Contiene la lista de los aviones que se están simulando con los parámetros de cada uno de ellos, se actualiza constantemente según avanza la simulación.
  - Retrasoavion: Lista de retrasos tipo 2 y tipo 3 que aplicará el agente durante la simulación

  - Retrasotiempo: Lista de retrasos tipo 1 que aplicará el agente durante la simulación
  - Aerolíneas: lista de aerolíneas con agente propio
- **Comportamientos**:
  - Como existen agentes AO específicos para una aerolínea, este agente tiene que identificar en cada caso que aviones son suyos y que aviones no. Por lo que al leer el documento VuelosAerolinea.txt no recopilará toda la información del documento, sino que cada agente guardará únicamente la información de los aviones que le corresponden.

4.3.1.5 Ground Handling (GH)

- **Descripción:** Simula los comportamientos de las empresas de Handling en el aeropuerto. Se encarga de controlar que los tiempos de turn around se cumplen y recibe estos datos de una base de datos de tiempos por modelo y aerolínea.
  Encontraremos más de un agente GH en el programa ya que cada uno de ellos representa a una empresa de Handling. La manera de relacionar los aviones de las diferentes aerolíneas con los agentes GH es a través de una serie de contratos de servicios que se introducen como parámetros iniciales. También encontramos un agente GH-GEN para que administre aviones sin agente GH propio.
- **Archivos**: El agente GH no leerá ningún documento
- **Ejecuta los milestones**:
  - Milestone 8 → ACGT
- **Tiempos definidos**: El GH define por primera vez el tiempo de:
  - ACGT→ Lo define cuando conoce el tiempo de EIBT
  - ASBT → Lo define cuando conoce el tiempo de ACGT
- **Define/Actualiza los milestones:**
  - Milestone 8 (ACGT) → cuando se ha actualizado EIBT
  - Milestone 11(ASBT) → cuando se ha actualizado ACGT
  - Milestone 15 (EOBT) → cuando el tiempo de Turn Around es muy pequeño
- **Actualizaciones de tiempos**:
  - ACGT → cuando se ha actualizado EIBT
  - ASBT → cuando se ha actualizado ACGT
  - EOBT → cuando el tiempo de Turn Around es muy pequeño
- **Constantes**:
  - avionesSimulacion: entero que indica la cantidad de aviones que se simularán

  - EIBTtoACGT: Minutos necesarios entre EIBT y ACGT
  - ACGTtoASBT: Minutos necesarios entre ACGT y ASBT
- **Parámetros**:
  - Tabla: Contiene la lista de los aviones que se están simulando con los parámetros de cada uno de ellos, se actualiza constantemente según avanza la simulación.
  - Retrasoavion: Lista de retrasos tipo 2 y tipo 3 que aplicará el agente durante la simulación
  - Retrasotiempo: Lista de retrasos tipo 1 que aplicará el agente durante la simulación
  - AvionesInicio: Lista de mensajes recibidos que se tienen de procesar.
- **Comportamientos**:
  - Este agente es el único que se comunica con un agente externo para solicitar información. Se comunica con el agente TunrAround, y le pregunta por información sobre el tiempo de turn around mínimo que necesitará un avión en concreto. Para realizar esto, el agente envía un mensaje y espera a que el TurnAround responda con la respuesta.
  - Relacionada con la tarea de preguntar al agente externo por el tiempo de turn around, queda claro que el GH esperará una respuesta. Pero sucederá que mientras espera esta respuesta algún otro agente le envía un mensaje con información para procesar. Esto sería un problema, pues no puede procesar ese mensaje hasta que no termine de procesar sus cálculos actuales. Por eso se ha creado una lista de mensajes en espera (AvionesInicio), donde se guardarán los mensajes que llegan mientras el agente está esperando respuestas del TurnAround.

4.3.2 Agentes Simulación CDM

4.3.2.1 Information Sharing

- **Descripción:** Este agente representa el programa a través del cual los diferentes *stakeholders* del aeropuerto comparten la información de los aviones. Se encarga de recibir actualizaciones e informar a todos los agentes de estas, con el objetivo de que todos tengan la misma información en todo momento.
- **Archivos**: El agente IS no leerá ningún documento
- **Comportamientos:** Este agente tiene dos comportamientos que actuarán en paralelo:

  - Comunicaciones: Este primer comportamiento únicamente se encarga de recibir los mensajes que recibe el agente y guardarlos en una lista de manera ordenada para que el otro agente los vaya procesando uno a uno. El único mensaje que se quedará este comportamiento es el que contiene los parámetros de simulación, cuando llegue este mensaje lo procesará y se guardará la información en la memoria del agente.
  - AdministrarTabla: Es el comportamiento principal del IS. Este comportamiento leerá y decodificará los mensajes facilitados por el comportamiento Comunicaciones. Cada uno de estos mensajes contendrá información sobre un avión que ha actualizado alguno de sus parámetros. Este agente se guardará la información sobre estas nuevas actualizaciones y realizará un broadcast a todos los agentes para que estos estén también actualizados en todo momento.
- **Constantes**:
  - avionesSimulacion: entero que indica la cantidad de aviones que se simularán
  - CDMAgents: lista de agentes que recibirá los mensajes con las actualizaciones
  - AOforGH: contiene los contratos de las empresas de GH con agente propio en el programa e información sobre las aerolíneas con las que trabajan cada uno de ellos.
  - Aerolíneas: lista de aerolíneas con agente propio
- **Parámetros**:
  - Tabla: Contiene la lista de los aviones que se están simulando con los parámetros de cada uno de ellos, se actualiza constantemente según avanza la simulación.
  - ListaMensajes: Lista con los mensajes que aún tiene que procesar el agente IS
- **Comportamientos**:
  - Este agente no guarda el avión directamente a la tabla, sino que en todo momento mira que valores han sido actualizados en los parámetros del avión y solo actualiza esos parámetros a los del avión que ya tenía guardado en la tabla
  - Después de resultar informado de cualquier actualización el agente realizará un broadcast a todos los agentes. En cualquier broadcast tendrá de buscar que agente AO y GH es el que espera recibir esa información.

4.3.2.2 **Milestone Trigger**

- **Descripción:** El milestone trigger es un agente que representa el paso del tiempo en la simulación. Contiene un timer y ejecuta los diferentes milestones de la simulación en orden cronológico. Cuando detecta que se han acabado los milestones ejecuta el procedimiento de finalizar el programa.

  Además de ejecutarlos, tiene que recibir constantemente actualizaciones de estos milestones y ordenarlos cronologicamente para actualizar su lista.
- **Archivos**: El agente Milestone Trigger leerá el documento de TriggerCDM.txt que contendrá información sobre los primeros milestones de los aviones, luego ira recibiendo mensajes con avisos de nuevos tiempos de milestones para terminar de actualizar la lista con los milestones que sucederán durante la simulación. El agente MilestoneTrigger será el encargado de escribir el documento MilestonesResultados.txt con una lista de todos los milestones que se han realizado durante la simulación, ordenados cronológicamente.
- **Comportamientos:** Este agente tiene dos comportamientos que actuarán en paralelo:
  - AdministrarMilestones: Este comportamiento es el encargado de recibir los mensajes con nuevos milestones o con las actualizaciones de los tiempos de los milestones ya existentes. Al recibir cualquiera de estos mensajes el comportamiento guardará la información de estos milestones en la lista y la mantendrá siempre ordenada temporalmente.

    Este comportamiento también guardará la información con los parámetros de la simulación.

    El agente AdministrarMilestones también será el encargado de leer el mensaje que avisa de que se acaba la simulación en caso de que el usuario decida parar la simulación prematuramente mediante la interfaz.

    En el caso de recibir el mensaje avisando de que se tiene que empezar la simulación pasará esta información al otro comportamiento del agente.
  - TriggerMilestones: Es el comportamiento que avisa a los agentes de que suceden los milestones, y por lo tanto es el agente que hace que la simulación avance en el tiempo mediante el aviso de milestones realizados. Este agente leerá el documento TriggerCDM.txt cuando se le avise de que puede empezar la simulación. Posteriormente activará el timer.

A cada tic del timer este agente avisará de que un avión ha realizado un milestone. El milestone del que avisa cada vez lo coge de la lista de milestones que administra el otro comportamiento del agente (comportamiento AdministrarMilesontes). Cada vez mirará que agente es el encargado de ejecutar ese milestone y mandará el aviso de que se realiza el milestone al agente correspondiente.

- **Constantes**:
  - avionesSimulacion: entero que indica la cantidad de aviones que se simularán
  - AOforGH: contiene los contratos de las empresas de GH con agente propio en el programa e información sobre las aerolíneas con las que trabajan cada uno de ellos.
  - Aerolíneas: lista de aerolíneas con agente propio
  - ruta: Ruta donde leer el documento TriggerCDM.txt
  - rutaMilestones: Ruta donde guardar los milestones que suceden, MilestonesResultados.txt
  - VelocidadSimulacion: tiempo en milisegundos que pasa entre cada tic del timer.
- **Parámetros**:
  - Tabla: Contiene la lista de los aviones que se están simulando con los parámetros de cada uno de ellos, se actualiza constantemente según avanza la simulación.
  - MensajeInicio: Lista con los mensajes que aún tiene que procesar el agente MilestoneTrigger
  - Milestonelist: Lista de milestones que sucederán ordenados cronológicamente. El agente AdministrarMilestones está actualizando esta lista constantemente según las actualizaciones., el agente TriggerMilestones usa esta lista para saber de qué milestones tiene que avisar.
  - OldMilestoneList: Lista de milestones que ya han sucedido. Esta lista es la que se usará para pintar el documento MilestonesResultados.txt
- **Comportamientos**:
  - Este agente es el único que utiliza un timer. Dentro del timer se ha programado para que envíe el aviso de que un avión ha realizado un milestone, esto es lo que permite al programa simular que avanza en el tiempo.
  - Cada vez que avise de que un avión ha superado un milestone, tendrá de buscar el agente encargado de recibir esa información

en función de la aerolínea del avión y del milestone que este superando.

4.3.3 Agentes Software Input

4.3.3.1 Simulator Input

- **Descripción:** Este agente se encarga de preparar la simulación. Recibe todos los parámetros iniciales a través de la interfaz y de cargar los archivos, genera los archivos necesarios para la simulación, e inicia la simulación cuando el usuario lo desea.
  También da diferentes opciones al usuario, como administrar los retrasos de la simualción o saltarse la simulación cargando directamente un documento de resultados externo.
- **Archivos**: El agente Simulator es el encargado de leer los siguientes documentos:
  - Input.txt: Documento con toda la información de los aviones que se simularán en cada ocasión. Este documento lo ha escrito el DDR2 anteriormente.
  - Retrasos.txt: Leerá también el documento de retrasos para informar a los otros agentes de los retrasos que tendrán de aplicar en cada ocasión. Sólo leerá este documento si el usuario ha seleccionado esta opción mediante la interfaz.
  - Airport.txt: Leerá el documento Airport.txt con los parámetros de simulación.

  Este agente también escribirá una serie de documentos:
  - FlightPlans.txt: Documento con la información de los planes de vuelo de los aviones que llegan al aeropuerto
  - FlightPlansSalidas.txt: Documento con la información de los planes de vuelo de los aviones que salen del aeropuerto.
  - TriggerCDM: Documento con los primeros milestones que realizarán los aviones, información extraída de los planes de vuelos de estos.
  - AO1: Contiene la información sobre las horas de aterrizaje de los aviones
  - AO2: Contiene la información sobre las horas de despegue de los aviones
  - VuelosAerolinea: Contiene información sobre las horas de llegada y salida de los aviones.
  - Retrasos: Aunque este agente también leerá el documento Retrasos.txt el mismo es el encargado de escribirlo. Escribirá el

documento con la información solicitada por el usuario mediante las interfaces.

- **Comportamientos:** Este agente tiene un único comportamiento:
  - RecepcionAvion: Este comportamiento es el primero que actuará en el programa. Esperará a que el usuario mediante la interfaz decida que función del programa quiere ejecutar. Este comportamiento se comunicará con el DDR2 para que lea el documento de Eurocontrol, también será el encargado de crear los documentos input que necesitarán los otros agentes.

    Este comportamiento será el encargado de avisar a todos los agentes con los parámetros seleccionados para la simulación. Y será el que organizará las interfaces de inicialización y de retrasos para que se abran cuando les corresponde.

    También será el encargado de mandar el mensaje que iniciara la simulación.
- **Constantes**:
  - rutaInput: Ruta donde obtener el documento input.txt
  - rutaFP: Ruta donde se guardará el documento FlightPlans.txt
  - rutaFPS: Ruta donde se guardará el documento FlightPlansSalidas.txt
  - rutaMilestones: Ruta donde guardar los milestones que ya se conocen de los planos de vuelo, MilestonesResultados.txt.
  - rutaAO1: Ruta donde guardar el documento AO1.txt
  - rutaAO2: Ruta donde guardar el documento AO2.txt
  - rutaAO: Ruta donde guardar el documento AO.txt
  - rutaRetrasos: ruta del documento donde guardará y leerá la información relacionada con los retrasos.
  - AOforGH: contiene los contratos de las empresas de GH con agente propio en el programa e información sobre las aerolíneas con las que trabajan cada uno de ellos.
  - Aerolíneas: lista de aerolíneas con agente propio
- **Parámetros**:
  - countArr: contador de aviones que llegan al aeropuerto
  - countDep: contador de aviones que salen del aeropuerto
  - DDR2valido: indica si ya se ha leído el documento de Eurocontrol y se ha creado el documento input.txt
  - Retrasos: Lista de retrasos que se aplicarán
- **Comportamientos**:
  - Comprobación constante de las selecciones desde interfaz. Tiene que estar atento a las decisiones que toma el usuario desde las

interfaces, por este motivo está constantemente preguntando a las interfaces por las decisiones que toma el usuario.

- Tiene que crear los documentos en 2 fases, pues algunos de ellos requieren la información de los retrasos que se aplicarán y puede ser que aún no hayan sido definidos por el usuario. Por este motivo se crean primero los documentos que no requieren esta información y posteriormente cuando ya están definidos los retrasos se crean los otros documentos.

- **Interfaces**:
  - **A-CDM Simulator Input**

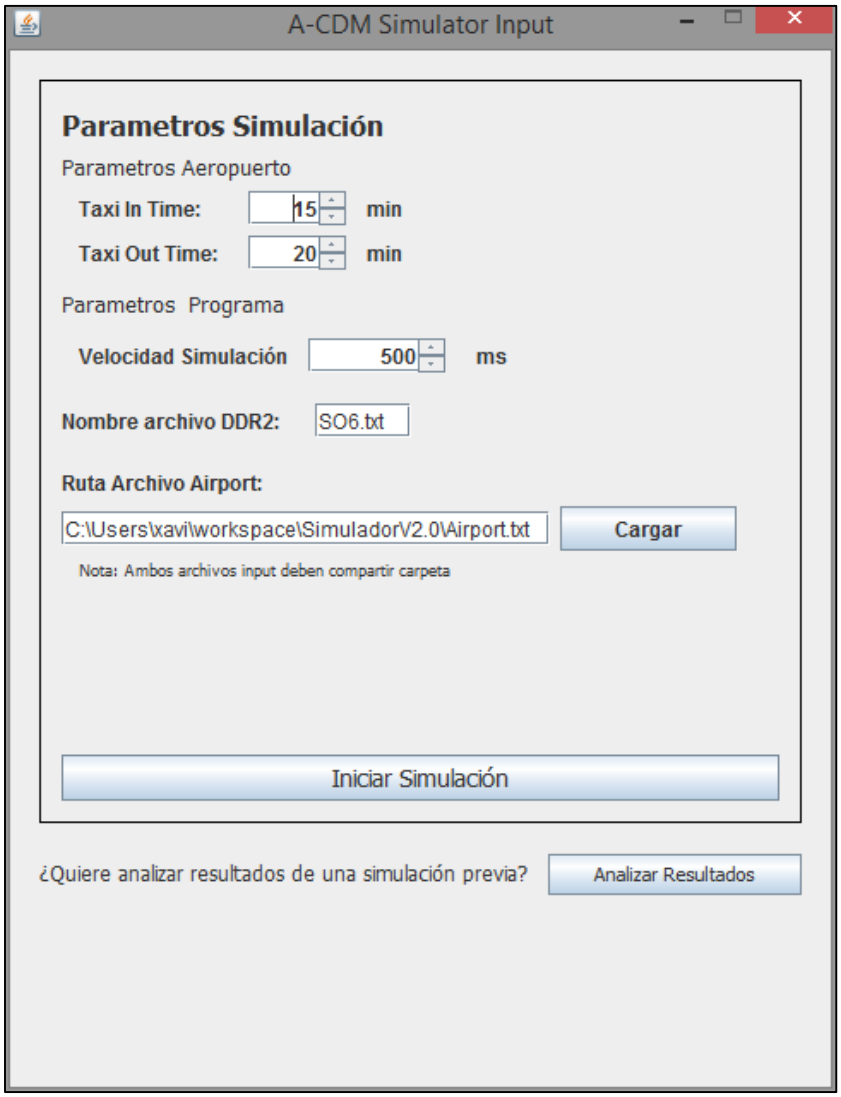


Figura 4-11: Interfaz A-CDM Simulator Input

Primera Interfaz que encontraremos en el programa, desde aquí podemos escoger los parámetros de simulación deseados, y los documentos de entrada que deseamos simular. El programa realizará una comprobación

de que los documentos existan y estén escritos con el formato correcto. Desde esta interfaz podremos decidir si queremos implementar retrasos a la simulación o si nuestras intenciones no pasan por realizar una nueva simulación sino que únicamente queremos visualizar los resultados de una simulación ya realizada en otro momento.

- **Retrasos Simulación**

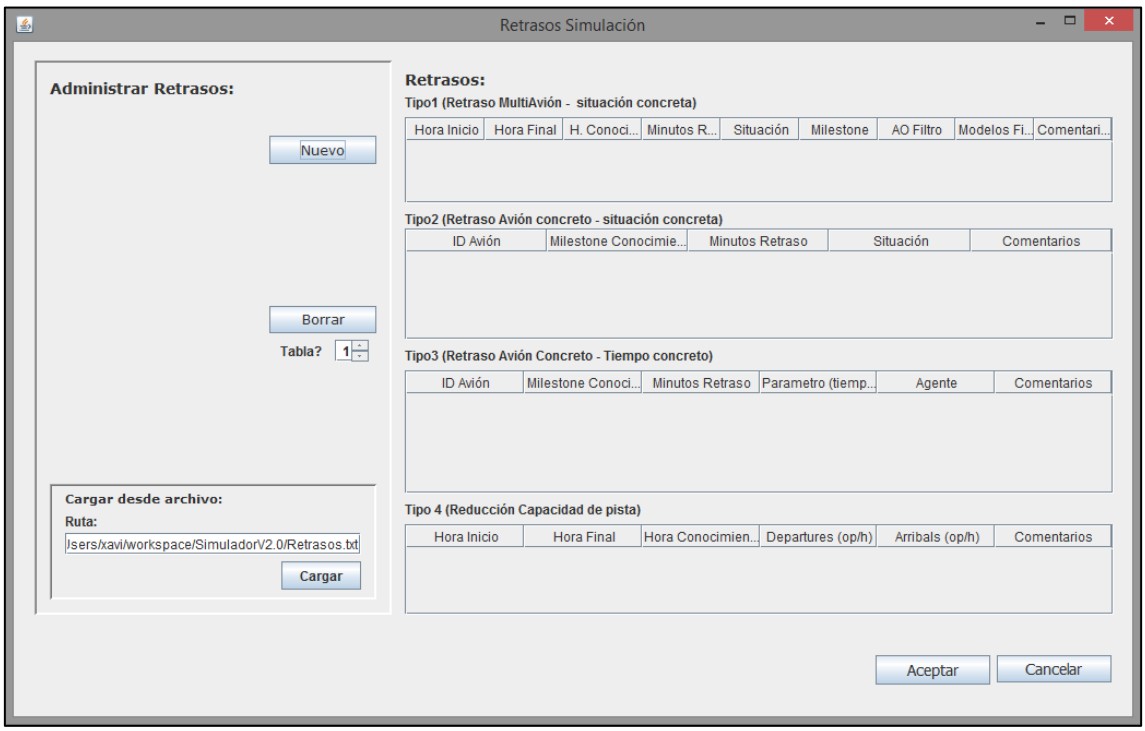


Figura 4-12: Interfaz Retrasos Simulación

En esta interfaz visualizaremos los retrasos que estamos a punto de aplicar a nuestra simulación, separados en 4 tablas según el tipo de retraso de cada uno. Podemos crearlos uno a uno (botón nuevo) y también podemos decidir cargar todos los retrasos que encuentre en un documento externo. El programa comprobará que la ruta facilitada sea verídica. También da la opción al usuario de eliminar un retraso definido previamente, únicamente tenemos de seleccionar en la propia tabla el retraso que queremos eliminar, avisar al programa de que tabla es la que queremos modificar (mediante un spinner) y acto seguido pulsar el botón borrar. Si no realizamos estos pasos el programa no actuará y nos avisará.

- **Formulario Retrasos**

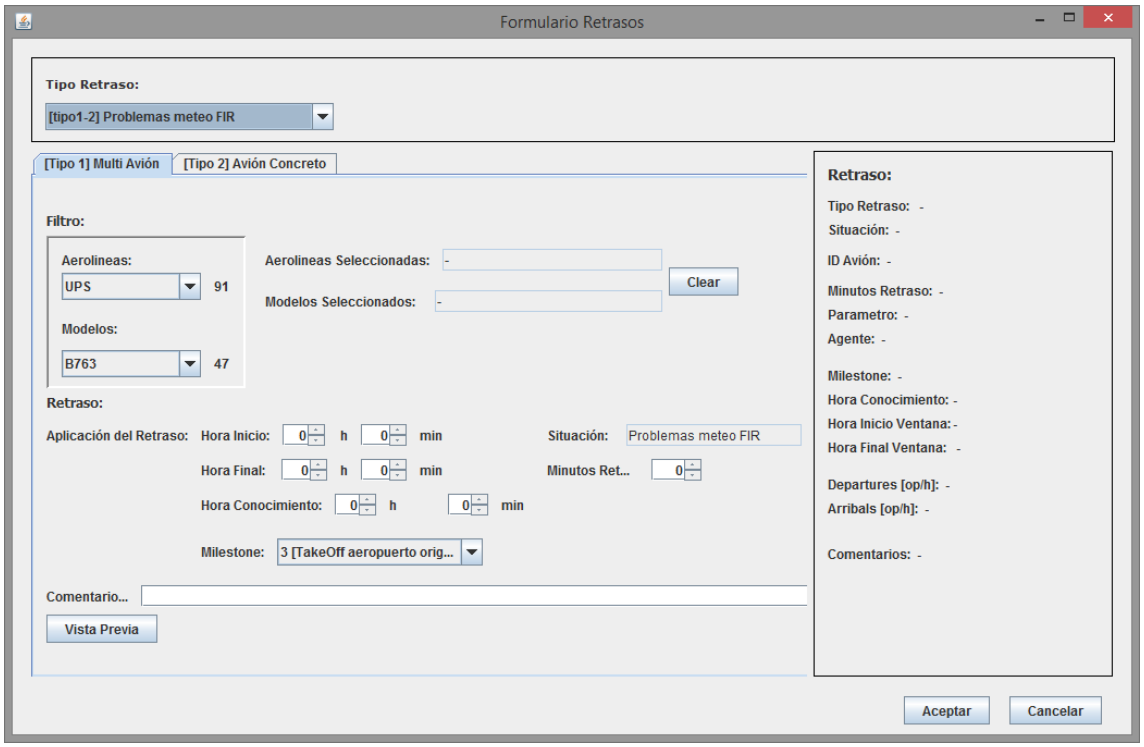


Figura 4-13: Interfaz Formulario Retrasos

Desde esta interfaz podremos crear y definir un nuevo retraso con los parámetros que deseemos. Primero seleccionaremos el tipo de retraso que queremos seleccionar (tipo 1, tipo 2, tipo 3 o tipo 4). Según la opción seleccionada la interfaz nos preguntará por unos parámetros u otros. Una vez respondidos a todos ellos correctamente, diremos al programa que cree una vista previa del retraso, en este momento el programa realizará un escaneado de nuestras respuestas para asegurar de que todas ellas están escritas correctamente, por ejemplo no ha de dejar que el tiempo de finalizar un retraso sea antes que el tiempo de iniciar este retraso, ni ha de permitir que un avión en el milestone 15 aplique un retraso en el tiempo del milestone 10, pues ese milestone ya ha sido superado. Si el retraso se ha definido correctamente con todos sus parámetros podremos seleccionar aceptar y de esta manera quedará definido y a punto para ser aplicado en la simulación.

- **Retrasos Simulación**

En esta interfaz tenemos la opción de realizar una copia de seguridad de los retrasos que estamos a punto de implementar. Simplemente tenemos de decidir si la queremos o no. El programa en cualquier caso guardará un documento con la información de los retrasos en el workspace bajo el nombre de Retrasos.txt, desde esta interfaz podemos crear una segunda copia.

4.3.3.2 **DDR2**

Descripción:
El agente DDR2, solo actuará una vez a lo largo de cada simulación, su función consiste en transformar el documento de entrada con la información SO6 extraída de Eurocontrol, a un documento con la estructura adecuada para que el resto de los agentes puedan entender la información de Eurocontrol. Por otro lado, el documento DDR2 puede contener información sobre todos los vuelos europeos, según los filtros seleccionados en el momento de obtener dicho documento, este agente también se encargara de separar los vuelos que no afectan a nuestro aeropuerto.

El agente DDR2 también será el encargado de llevar a cabo la tarea de mirar las relaciones entre los aviones que aterrizan y despegan, para poder suponer en qué casos un vuelo de llegada y un vuelo de salida corresponden a un mismo avión.

Actuación:

- **Objetivo**

A continuación explicaremos más detalladamente todo el proceso que realiza el agente DDR2 para crear el documento Input.txt que utilizará el Simulator Input:

Recordemos que el documento de entrada que necesita el agente SimulatorInput ha de contener una línea para cada llegada (tabla 4-1) y otra línea para cada salida (tabla 4-2), cuando el valor del callsign de una llegada y una salida coincida significa que el avión que entra es el mismo que se va más tarde.

Llegada	*Callsign*	*Modelo*	*Aeropuerto Out*	*Aeropuerto In*	*ETOT'*	*TFIR*	*TAPP*	*ELDT*
1	IBE450	A321	LEPA	LEBL	0910	0946	0951	0956

Tabla 4-1: Estructura de una llegada en el documento creado por DDR2

Salida	*Callsign*	*Modelo*	*Aeropuerto Out*	*Aeropuerto In*	*ETOT*	*TFIR 0*
2	IBE450	A321	LEBL	LEPA	1121	IBE46

Tabla 4-2: Estructura de una salida en el documento creado por DDR2

En cada línea la información de una llegada o una salida está organizada tal y como se muestra en las tablas anteriores.

- **Recepción aviso inicio lectura documento de Eurocontrol**

El agente DDR2 entrara en acción al recibir un mensaje proveniente del agente SimulatorInput. Este mensaje contiene, como es habitual, la información en una clase Aircraft. Este Aircraft, viene prácticamente vacío, solo contendrá información en el apartado de incidencias. El apartado incidencias estará formado por la información de la tabla 4-3:

Nombre documento SO6	Ruta carpeta documentos de entrada	Lista WayPoints FIR	Lista WayPoints TAPP	Código ICAO del aeropuerto a estudiar

Tabla 4-3: Información que recibe el agente DDR2

La lista WayPoints FIR contiene una lista con todos los WayPoints que rodean la FIR del aeropuerto estudiado, mientras que la lista WayPoints TAPP contiene una lista con todos los WayPoints desde los que un avión puede empezar la aproximación.

Una vez recibido este mensaje el agente empezara a trabajar. El comportamiento "action" se desbloqueara debido a que ha recibido un mensaje y empezará con los cálculos. Primero de todo extraerá la información del Aircraft. A continuación procederá con la lectura del documento DDR2 de Eurocontrol para proceder a escribir el documento Input.txt.

- **Lectura documento de Eurocontrol**

Para realizar esta tarea el programa empieza leyendo línea a línea la información del documento DDR2, separando en primera instancia los aviones que llegan o salen del aeropuerto que estamos estudiando. Si ni el aeropuerto de destino ni el de salida coinciden con el aeropuerto estudiado, significa que ese avión no es de nuestro interés y el programa pasará a la siguiente línea.

SALIDAS:

En caso de que el aeropuerto de origen coincida con nuestro aeropuerto, significara que ese avión es una salida, y recogeremos los siguientes datos:

Callsign – Modelo - Aeropuerto OUT - Aeropuerto IN - ETOT

El ETOT es la hora a la que empieza el primer segmento del vuelo, pues es el momento en que sale del aeropuerto.

LLEGADAS:

En caso de que el aeropuerto de destino coincida con nuestro aeropuerto, significara que ese avión es una llegada, y recogeremos los siguientes datos:

Callsign – Modelo - Aeropuerto OUT - Aeropuerto IN - ETOT' – TFIR – TAPP - ELDT

- ETOT' es la hora en la que sale del aeropuerto de origen de este vuelo.
- TFIR se calcula mirando los WayPoints a cada segmento del vuelo, para ver cuando el avión pasa por uno de los WayPoints que conforman las FIR del aeropuerto estudiado.
- TAPP se calcula mirando los WayPoints a cada segmento del vuelo, para ver cuando el avión pasa por uno de los WayPoints que conforman los puntos desde donde se puede empezar la aproximación a pista.
- ELDT es el tiempo en el que finaliza el último segmento del vuelo, pues es el momento en el que el avión llegará a la pista.

Con las llegadas podría darse el caso de que el avión no pase por ninguno de los WP definidos para la FIR, igual que podría no pasar por ningún WP de los definidos para empezar las aproximaciones, entonces el programa asignará por defecto TFIR=ELDT-10min y para la aproximación TAPP=ELDT-5min.

Podría darse el caso de que un avión saliera a altas horas de la tarde-noche, y que por eso llegara pasadas las 12 de la noche a nuestro aeropuerto, en ese caso el programa sumara 24h a los tiempos que sucedan en el siguiente día de la simulación. Por lo que si ELDT sucede a las 0200 (HHMM), el programa lo considerará como 2600(HHMM).

En caso de que el avión despegue a primera hora de la mañana podría darse el caso que el primer milestone (3 horas antes del despegue), suceda en un día distinto al de la simulación, en este caso consideraríamos la hora de milestone 1 como un valor negativo, pudiéndose dar el caso que el milestone 1 suceda a las -0300(HHMM).

También tener en cuenta que en ciertos casos podría ocurrir que la lista de vuelos tenga 2 vuelos distintos con el mismo callsign, entonces el programa añade una "/" detrás del callsign del segundo que encuentra para distinguirlo del primero. Si se diera el caso de que llegan o salen de nuestro aeropuerto más de dos aviones con el mismo callsign, el primero tendrá una "/", el segundo dos "//", y así consecutivamente añadiendo una / por cada vez que el mismo callsign aparezca para vuelos distintos. Por ejemplo si aparecieran 3 IBE340 tendríamos: IBE340, IBE340/, IBE340//.

- **Relacionar llegadas y salidas**

Ahora el programa ya ha detectado todas las llegadas y salidas que estudiaremos. La siguiente tarea del agente DDR2 será relacionar las llegadas con las salidas, para saber cuándo es un mismo avión el que entra y el que sale del aeropuerto.

Para realizar esta tarea, mirará en la lista de despegues avión por avión comprobando si existe un avión de la misma compañía y mismo modelo, que aterrizara en el aeropuerto previamente con una diferencia de tiempo entre la salida y la llegada superior a 35 minutos, de esta manera se podrá considerar que el avión que aterriza, realiza el ground handling y despega de nuevo para otro destino.

Cuando estas condiciones se cumplan consideraremos que existe una relación entre una llegada y una salida, pero los callsigns de los dos vuelos no siempre coincidirán, por lo que para no perder información, el programa variará la información del vuelo de salida:

2 – Callsign (llegada) – Modelo - Aeropuerto OUT - Aeropuerto IN – ETOT – Callsign (Salida)

El hecho de identificar los 2 callsigns en la misma línea es porque el programa no detectaría que tenemos un mismo avión con 2 Callsign distintos. Pues el

agente Simulator Input considera que los aviones con un mismo callsign son siempre el mismo avión. Por esto si el agente DDR2 quiere relacionar 2 vuelos (considerar que los realiza un mismo avión) lo que tendrá de hacer es modificar el callsign del vuelo que sale, de manera que coincida con el vuelo que se está relacionando.

El programa al encontrar dos entradas que por tiempos, modelo y aerolínea podrían coincidir con un despegue, considerará siempre que el avión que lleva más tiempo esperando en el aeropuerto es el mismo que realizará el despegue. De la misma manera nunca relacionará una llegada con 2 salidas, pues como es lógico un avión solo puede realizar una salida por cada llegada. De la misma manera tampoco relacionaremos 2 llegadas con una salida.

- **Escritura Input.txt**

Una vez buscadas todas las relaciones, veremos que tendremos, aviones que solo llegan, aviones que solo salen y aviones que realizan todo el proceso y llegan al aeropuerto y posteriormente despegan para ir a otro destino.

A continuación el agente escribirá en el fichero input.txt la información correspondiente a cada vuelo que aterriza o despega, y por último mandara un mensaje al agente SimulatorInput para informar de que el documento se ha creado con éxito, o de que ha habido algún problema por lo que no se ha creado dicho documento. Los problemas habituales son que la información sobre la ruta donde está guardado el documento de entrada DDR2 es erróneo, o que dicho documento ha sido manipulado, causando un error en la lectura

4.3.4 Agentes Software Output

4.3.4.1 Simulador Gráfica

- **Descripción:** Tiene la función de mostrar al usuario como está avanzando la simulación en el tiempo. Este agente es únicamente informativo, no realiza ningún cálculo, se limita a escuchar los mensajes de los otros agentes y a mostrar los datos por pantalla en una tabla para que resulte fácil de entender para el usuario. También da la opción al usuario mediante la interfaz de decidir si quiere parar la simulación en cualquier momento.
- **Archivos**: El agente Simulador Gráfica no lee ni escribe ningún archivo.
- **Comportamientos:** Este agente tiene un único comportamiento:
    - RecepcionAvion: Recibe actualizaciones de parámetros de distintos aviones y se dedica a ir actualizando la tabla interna de aviones constantemente para mostrar por pantalla los datos actualizados en todo momento. Estará atento también a si el usuario decide realizar una simulación parcial seleccionando la opción de terminar la simulación antes de que el simulador llegue al final de los cálculos. En este caso este agente avisará a los otros

de que paren de simular, y se considerarán los resultados que tenga el programa en ese momento como los resultados finales de la simulación.

- **Constantes:**
  - avionesSimulacion: entero que indica la cantidad de aviones que se simularán
- **Parámetros**:
  - Tabla: Tabla donde se va guardando la información de los aviones que llegan al agente mediante mensajes de actualizaciones durante la simulación
- **Interfaces**:
  - **A-CDM Simulator**

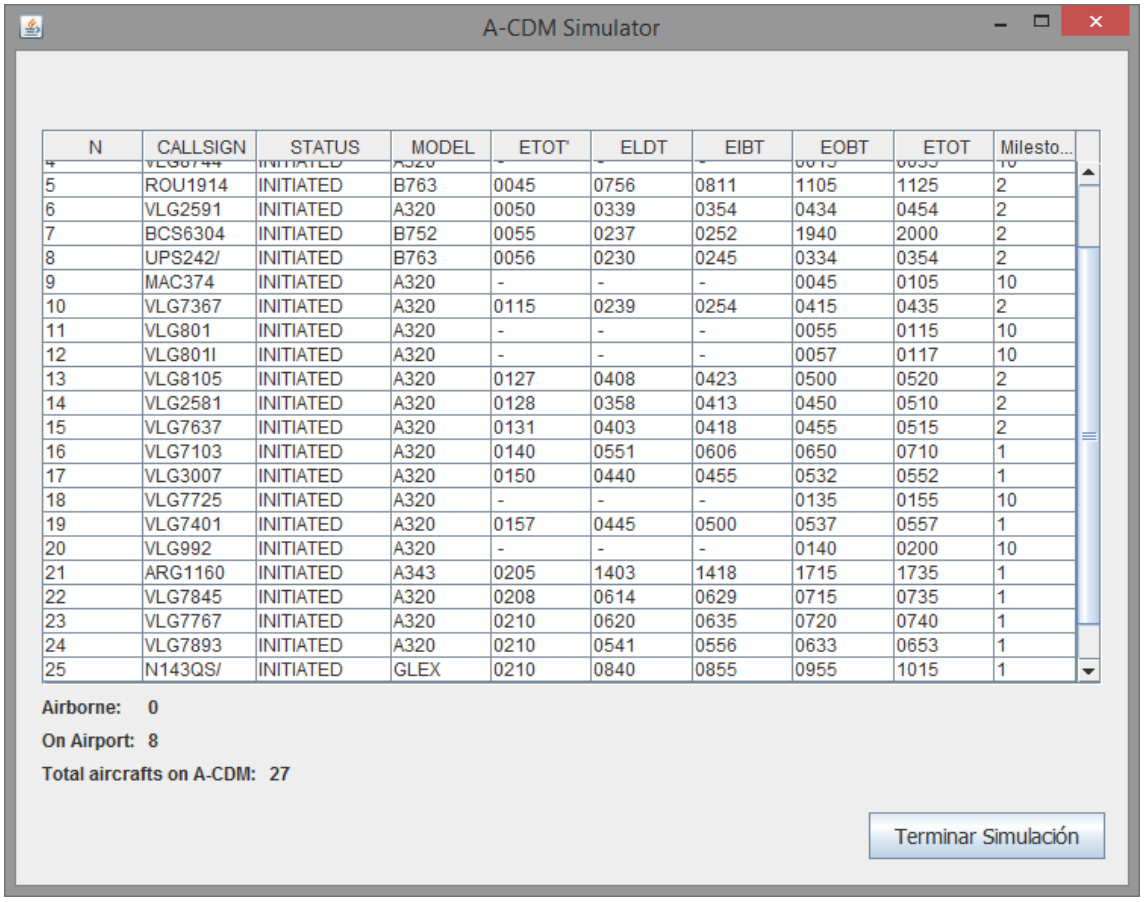


Figura 4-14: Interfaz A-CDM Simulator

Esta interfaz muestra en todo momento el estado de la simulación a tiempo real. El agente recibe un mensaje con una actualización de un avión y muestra por pantalla el estado de todos los aviones en todo momento. La interfaz realizará unos cálculos para determinar la cantidad de aviones en el A-CDM, en el aeropuerto y los que están aproximándose al aeropuerto.

4.3.4.2 **Output**

- **Descripción:** Tiene 3 funciones muy importantes, la primera es guardar los resultados de la simulación en un documento externo, la segunda es la de facilitar al usuario la lectura de los resultados por pantalla mediante una tabla de resultados, e incluso de comparar estos resultados con los realizados en otra simulación. Y por último permite al usuario realizar un pequeño análisis para poder extraer conclusiones de los resultados obtenidos. También dejará constancia de estos análisis en documentos de texto.
- **Archivos**: El agente Output es el encargado de escribir el documento results.txt con una lista de los aviones que se han simulado con los parámetros de estos al final de la simulación. A cada línea de documento la información de un avión.

  Si el usuario ha decidido desde la interfaz A-CDM Simulator Input cargar unos resultados en lugar de realizar una nueva simulación, este agente será el encargado de leer este documento con los resultados de una simulación realizada en otro momento.

  Si el usuario decide cargar una segunda tabla de resultados de otra simulación esta interfaz también leerá ese documento con la información de los aviones simulados en otra ocasión.

  Por último este agente escribirá los documentos de análisis de resultados donde mostrará los datos calculados al profundizar en los parámetros de los aviones al final de las simulaciones. Estos documentos solo se escribirán si el usuario desea analizar los resultados al terminar la simulación.
- **Comportamientos:** Este agente tiene un único comportamiento:
  - AvionEnTabla: Recibe actualizaciones de parámetros de distintos aviones y se dedica a ir actualizando la tabla interna de aviones constantemente. Si recibe el mensaje de fin de simulación escribirá el documento results.txt con los datos guardados en la tabla en ese momento y también mostrara en ese momento la tabla final con los resultados de la simulación.

    Si el usuario ha decidido que en lugar de realizar una simulación prefiere cargar unos resultados este agente leerá el documento y mostrará por pantalla los resultados que se calcularon en esa simulación.

    De la misma manera, si el usuario decide mediante interfaz cargar otro documento de resultados, este agente también leerá este documento y mostrará por pantalla 2 tablas a la vez de resultados.

- **Constantes**:
  - ruta: Ruta donde guardar los milestones que ya se conocen de los planos de vuelo, MilestonesResultados.txt.
  - avionesSimulacion: entero que indica la cantidad de aviones que se simularán
- **Parámetros**:
  - Tabla: Tabla donde se va guardando la información de los aviones que llegan al agente mediante mensajes de actualizaciones durante la simulación
  - tablaDoc: Tabla donde se guarda la información de los aviones extraídos del documento facilitado por el usuario mediante interfaz A-CDM Simulator Input
- **Interfaces**:
  - **A-CDM Resultados**

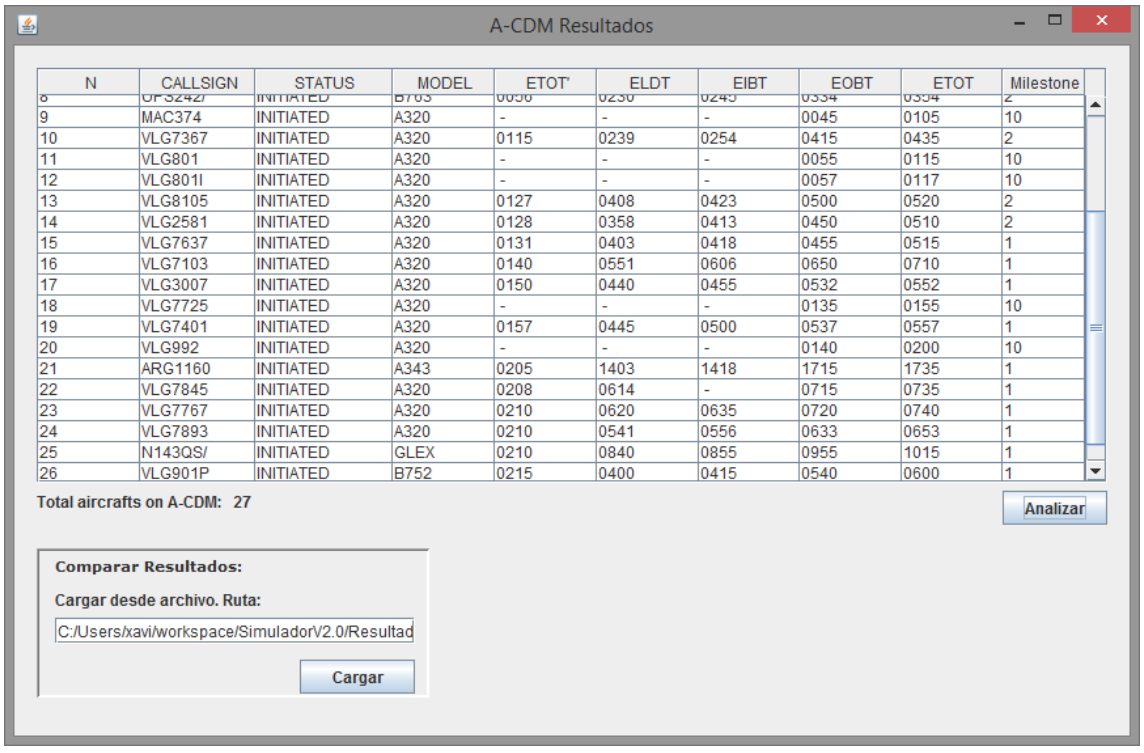


Figura 4-15: A-CDM Resultados

Esta interfaz muestra por pantalla la tabla de información de los aviones simulados durante la simulación o la tabla con los parámetros de los aviones que ha encontrado en el documento que el usuario le ha facilitado desde la interfaz A-CDM Simulator Input. Esta interfaz es únicamente informativa. Pero el usuario podrá entrar una nueva ruta a un documento de resultados para que el programa muestre una segunda tabla con los parámetros de los aviones de otra simulación con el objetivo de poder comparar los resultados de ambas simulaciones. El programa mirará que la ruta sea correcta y se pueda abrir el documento.

También se podrá seleccionar la opción Analizar, de esta manera abriremos la interfaz Análisis Resultados.

- **ACDM - Análisis**

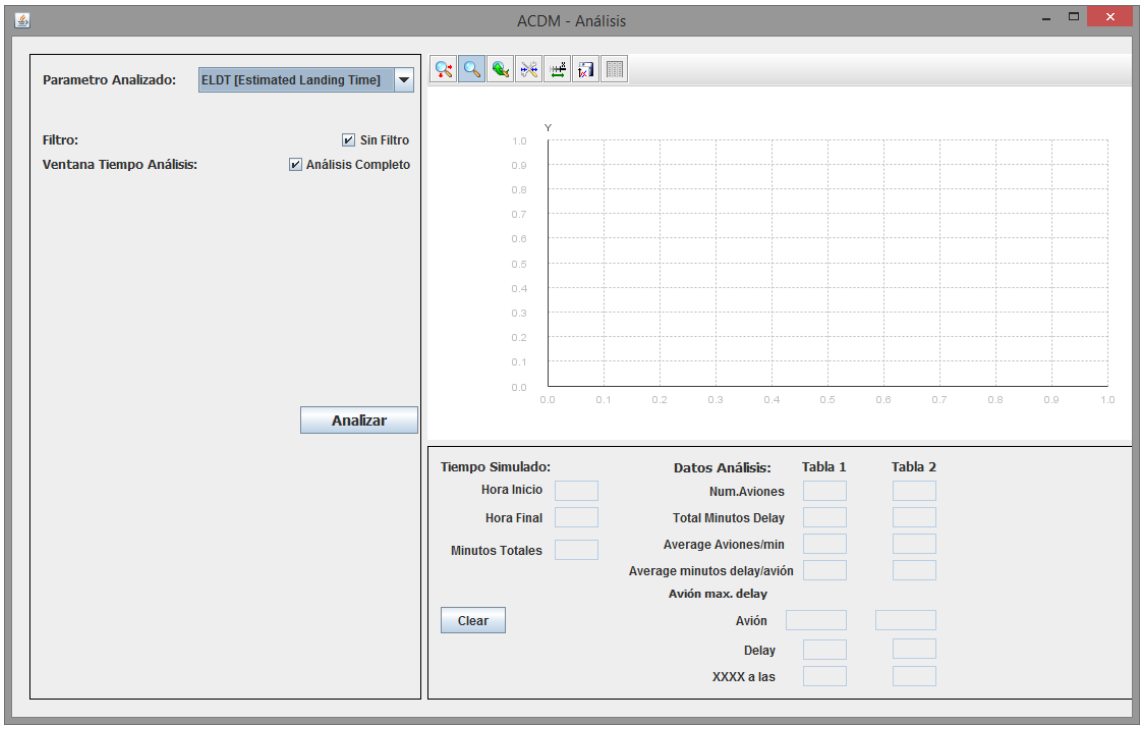


Figura 4-16: Interfaz ACDM – Análisis

Desde esta interfaz el agente Output realiza un análisis de los resultados encontrados durante la simulación. El análisis va centrado a estudiar de manera detallada los retrasos que sufren los aviones en uno de sus tiempos. Se escoge el parámetro a estudiar, con filtros o sin filtros, y se realiza un escaneado, avión por avión mirando si este avión tiene retraso, cuanto retraso, en que momento… Finalmente se muestran los valores por interfaz mediante TextBoxs y mediante una gráfica que muestra la cantidad de aviones que tendrían de haber superado ya ese milestone, pero por culpa de algún retraso aún no han llegado a ese punto. También escribirá los datos encontrados en este análisis en un documento de texto.

4.3.5 Agentes Externos

4.3.5.1 Turn Around

Descripción:

El agente Turn Around está creado como ejemplo para la creación de otros agentes externos. En un futuro si se desea mejorar el programa será necesaria la implementación de agentes externos, y el Turn Around es un ejemplo de un primer agente externo muy sencillo.

La razón principal para hacer este agente y no una simple explicación es que, aunque sus comportamientos internos son simples, la creación de un agente desde cero no es trivial y, sobretodo, era importante dejar un ejemplo claro de

cómo realizar la comunicación de un agente A-CDM (en este caso GH) con un agente externo al ACDM. Conseguir que esta comunicación no interfiera en el correcto funcionamiento del programa y, a la vez, cumpla su función eficientemente, se puede apreciar fácilmente en el código gracias a este ejemplo y a la siguiente explicación.

Actuación:

La función de este agente es informar al GH del tiempo que tarda en realizar el turn around un avión determinado siempre que el GH le pregunte. El tiempo de turn around es el tiempo que necesita un avión desde que realiza el EIBT hasta que está preparado para realizar el EOBT. La información con el tiempo necesario para realizar el turn around cada modelo de avión la recopilará del documento de entrada Turn-Around-Time.txt

Este agente tiene un comportamiento, *BuscarTurnAround* que está siempre esperando la llegada de nuevos mensajes provenientes del GH.

El primer mensaje que recibirá en cada simulación vendrá siempre del agente GH-GEN, con OpcionMensaje=3. Este mensaje será el de inicialización, y no requiere de respuesta. Será un mensaje que contendrá un objeto Aircraft y en las incidencias tendrá la información sobre los contratos de los diferentes GH que actúan en el simulador. Al recibir este mensaje también leerá el documento de Turn-Around-Time.txt y se guardara la información del tiempo de Turn Around de cada modelo y aerolínea. Si el documento de Turn-Around-Time.txt está mal escrito, o tiene otro nombre no aceptado o no existe, el agente externo considerará por defecto que todos los aviones sean del modelo que sea tardan 35min como mínimo para realizar el Turn Around. El valor de 35 minutos se ha decidido porque es el tiempo mínimo permitido para realizar la maniobra de turn around en el aeropuerto de Barcelona.

Una vez la simulación esta inicializada, este agente recibirá mensajes continuamente de cualquiera de los diferentes GH. En cada caso, mirara el modelo de avión recibido en el mensaje y la aerolínea, y buscará cual es el tiempo de Turn Around según la información que ha leído anteriormente del documento de entrada Turn-Around-Time.txt.

Si en la lista de tiempos de turn around no aparece ningún avión de ese modelo y aerolínea, procederá a buscar el tiempo de turn around para un avión de ese modelo, pero con aerolínea vacía, que se considera el caso general del modelo sin que importe la aerolínea del avión.

Cuando el modelo de avión de cual se ha solicitado el turn around no aparezca en la lista, se pueden dar 2 situaciones.

En primer lugar, que el documento leído anteriormente Turn-Around-Time.txt, contenga una línea para casos generales (por ejemplo: GEN 37), en este caso consideraríamos el valor al lado de “GEN” como el tiempo de Turn Around de ese modelo, añadiendo una constante en función de la categoría del avión siguiendo la tabla 4-4:

Categoría	*Turn Around Time*
1	GEN +15
2	GEN+10
3	GEN +5
4	GEN

Tabla 4-4: Tabla de tiempos Turn-Around caso General

El segundo caso se realiza cuando el modelo del avión del cual se solicita el turn around no está en el documento, y el documento de entrada tampoco facilita información del caso general "GEN". En ese caso el programa realizara el mismo proceso suponiendo GEN=35min. Dando lugar a la tabla 4-5:

Categoría	*Turn Around Time*
1	50 min
2	45 min
3	40 min
4	35 min

Tabla 4-5: Tiempos turn around en función de la categoría.

El proceso final que sigue el agente para definir el tiempo turn around en cada caso será el que muestra la figura 4-17:

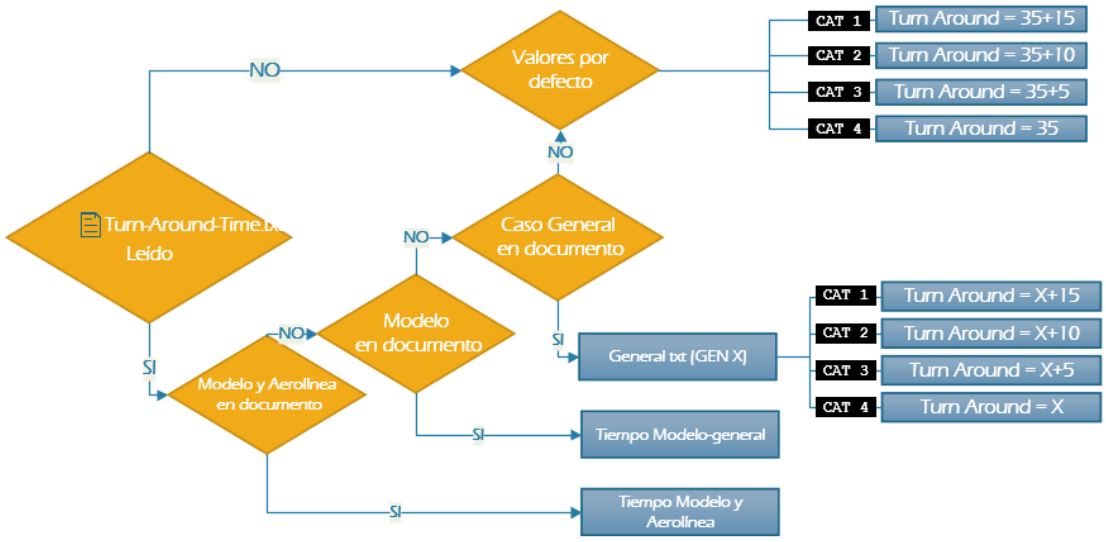


Figura 4-17: Proceso selección tiempo de turn around

Una vez tenemos un valor de turn around, miraremos los contratos de GH, y en función de la aerolínea del avión y los contratos el agente TurnAround decidirá a que agente GH mandará la información con el tiempo de turn around.

4.4 *Comunicaciones*

Una de las intenciones del proyecto es simular y analizar el flujo de comunicaciones que se realizan durante un A-CDM en un aeropuerto. Poder simular y monitorizar las comunicaciones es relativamente sencillo en los sistemas multiagente, esta es una de las razones de la elección de este tipo de sistemas.
Al tratarse de un programa multiagente, se basa en la comunicación entre los diferentes agentes, igual que pasa en el CDM de un aeropuerto. Esto hace necesario implementar a los agentes la capacidad de poder enviar mensajes, recibirlos y procesarlos de una manera asíncrona, es decir, que cada agente reaccione a los mensajes que recibe cuando los recibe, independientemente de la tasca que estén llevando los otros agentes.

Otra ventaja es poder rastrear los mensajes para saber qué información se mandan los agentes entre ellos, tanto para poder determinar el correcto funcionamiento del programa como para analizar las comunicaciones y sacar conclusiones finales sobre el ACDM real.

4.4.1 Como se envían los mensajes

Como ya se ha comentado en el apartado de JADE, este middleware nos proporciona una plataforma para poder enviar mensajes normalizados en las especificaciones FIPA

A la hora de programar estos mensajes se ha utilizado esta norma. Para adecuar al máximo las comunicaciones (lo que tenemos dentro de un mensaje) a la realidad de un A-CDM, algunos de los parámetros normalizados no se han utilizado, para simplificar las comunicaciones solo se han necesitado los parámetros de *Perfomative*, *Sender*, *Reciever* y *Content* [9] (en el cual se ha añadido toda la información del mensaje y del cual se habla más adelante en este apartado).

Entender la función de *Sender*, *Reciever* y *Content* en el mensaje es sencillo. Pero para entender *Performative* hay que saber que se refiere al “motivo” del mensaje, a la acción que realiza este mensaje. Este campo es obligatorio en este tipo de mensajes pero no es importante para el programa en sí, se ha utilizado en el programa para identificar rápidamente los diferentes tipos de mensajes en la representación gráfica de estos. Los valores de *Perfomative* utilizados en cada tipo de mensaje están especificados más adelante en este apartado.

Al mandar un mensaje debemos especificar quién es el destinatario, para ello añadiremos el AID del agente receptor del mensaje. Un AID debe contener el nombre del agente destinatario y la dirección. En nuestro caso la dirección no será necesaria pues será siempre la misma pues el programa se ejecuta siempre en el mismo host o máquina que el agente emisor.

Si observamos la figura 4-18, podemos ver un ejemplo de información contenida en un menaje. Si nos fijamos en *Receivers*, podemos ver que la línea que indica a quien se le manda el mensaje contiene incluso la información de la IP del ordenador, pero tal y como se ha explicado al trabajar siempre en un mismo ordenador no es necesario.

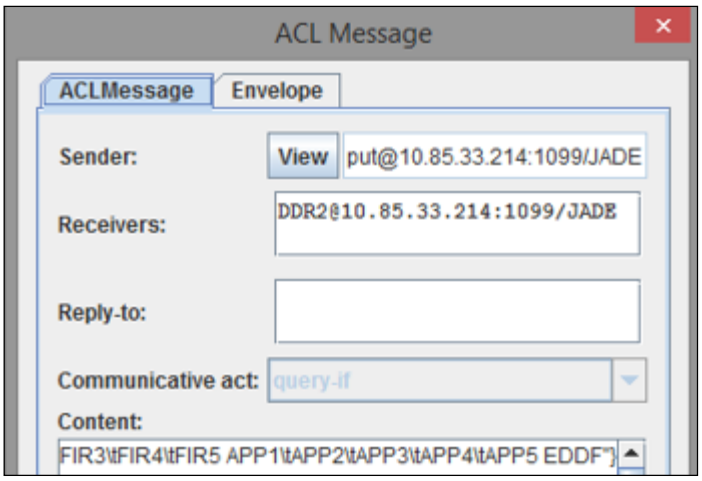


Figura 4-18: Información mensaje ACL

Podremos escribir las AID tal y como vemos en la figura 4-19, indicando que en cada caso se trata de una *LOCALNAME.*

```
AID MilestoneTrigger = new AID("MilestoneTrigger", AID.ISLOCALNAME);
AID CFMU = new AID("CFMU", AID.ISLOCALNAME);
AID DDR2 = new AID("DDR2", AID.ISLOCALNAME);
AID Output = new AID("Output", AID.ISLOCALNAME);
```

Figura 4-19: Definiendo Receivers para un mensaje

Cada mensaje coge un valor de *Perfomative* que dará una idea del objetivo de cada uno de los mensajes. Encontraremos mensajes informativos (Inform), solicitudes (Propose), peticiones (Query if) y propagación de informaciones (Propagate) [9].

Para mandar cualquier mensaje se escribirá el código de la figura 4-20:

```
// Definimos el tipo de mensaje FIPA que enviaremos (tiene que coincidir con el filtro de recepcion)
ACLMessage msg = new ACLMessage(ACLMessage.QUERY_IF);
msg.addReceiver(IS);

// Para enviar el avion en un mensaje, utilizamos las librerias Gson para convertir el objeto avion en un String.
Gson gson = new Gson();
String jsonOutput = gson.toJson(avion);

// Enviamos el avion
msg.setContent(jsonOutput);

send(msg);
```

Figura 4-20: Código de creación y envío de un mensaje ACL

En *msg.addReceiver* añadimos el AID del receptor del mensaje. En el caso de la imagen el receptor será el InfoSharing. En la figura 4-20 también podemos observar que el elemento que se mandará en el mensaje es de la clase Aircraft, en nuestro programa los mensajes entre agentes siempre contienen la información siguiendo la estructura de un objeto Aircraft, pues usualmente el mensaje será un Aircraft con algún tiempo o milestone actualizado. Por este

motivo se ha decidido que todos los agentes estarán únicamente preparados para mandar y recibir información de tipo Aircraft.

El proceso realizado para mandar un mensaje se basa en que el Json transforma un objeto Aircraft en un objeto String que contiene toda la información tenía el objeto Aircraft, y a continuación se procede a mandar esta información al destinatario.

El Json es una librería externa cuya función se basa en la transformación de una clase a un String, en nuestro caso el objeto Aircraft. Este String contendrá toda la información necesaria, para posteriormente poder ser transformado a la clase Aircraft de nuevo. Es necesario realizar esta transformación dado que al mandar un mensaje entre agentes, solo podemos mandar información de formato String. La librería Json también nos permite transformar el String recibido en el mensaje al objeto Aircraft con la información inicial.

Se ha decidido trabajar mediante la clase Aircraft, porque gracias a los atributos de esta clase, los agentes podrán enviar la información que necesitan. Tanto cuando se trate de avisos de hora, de solicitud de tiempos de turn around o de querer mandar toda la información relacionada a un avión para avisar de cualquier actualización en cualquier tiempo o dato del avión.

Los agentes una vez inicializados resultaran bloqueados a la espera de recibir un mensaje que procesar. El motivo de bloquear los agentes mientras esperan la información, es que mientras están bloqueados no se están ejecutando y por lo tanto no están consumiendo la CPU, por otra parte, es que podemos escoger los puntos exactos donde bloqueamos el programa a la espera de mensajes. Para bloquear un agente escribiremos el código de la figura 4-21:

```
//Bloquea el comportamiento hasta el momento en que el agente reciba un mensaje
ACLMessage mensaje = blockingReceive();
```

Figura 4-21: Código de bloqueo de un agente a la espera de un mensaje

Los comportamiento de un agente se ejecutan repetidamente, al finalizar las líneas de código del comportamiento vuelve a empezar el comportamiento por el principio de manera cíclica. Solo se parará en el momento de encontrar un blockingReceive como el de la figura 4-21, o en el momento en que se cierre el comportamiento.

Si un comportamiento encuentra un blockingReceive dejará de leer más líneas de código. En ese momento necesita la llegada de un mensaje para poder continuar con la ejecución del agente.

Todos los agentes que recibirán mensajes en algún momento de la simulación contienen al menos un blockingReceive, y seguidamente procederán realizando la extracción de la información que les llega en el mensaje. Ya sabemos que la información del mensaje siempre será un String que transformaremos a un objeto de clase Aircraft mediante el la librería Json.

Para extraer la información todos los agentes ejecutarán el código de la figura 4-22:

```
Gson gson = new Gson();
String jsonInput = msg.getContent();

Aircraft avion = new Aircraft();

avion = gson.fromJson(jsonInput, avion.getClass());
```

Figura 4-22: Código de extracción de información de un mensaje

Extraemos la información del mensaje en formato String, a continuación el Json procederá en transformar este formato al formato Aircraft que era el que teníamos inicialmente.

En la figura 4-23 podemos ver un ejemplo de conversación entre agentes. El agente SimulatorInput envía un primer mensaje al agente DDR2, el cual posteriormente le responde.

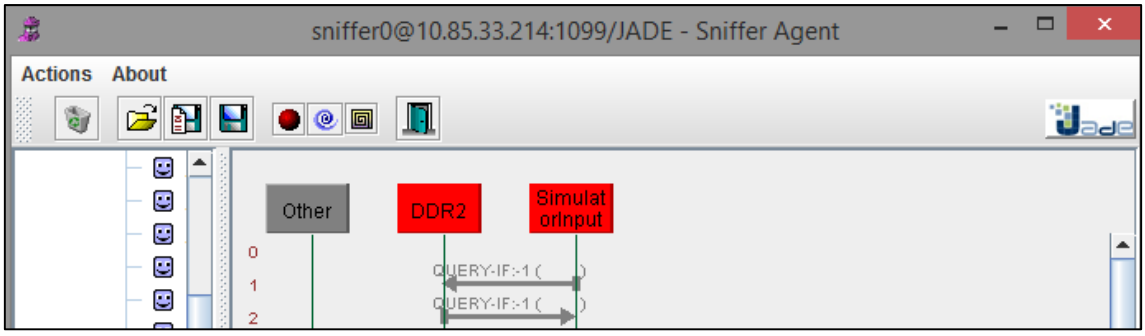


Figura 4-23: Captura de comunicaciones por Sniffer

En la imagen 4-24 podemos ver un ejemplo de la información que contiene un mensaje. En *Sender* encontramos la información relacionada al agente que manda el mensaje y en *Receivers* la información sobre el agente o agentes que reciben el mensaje. La información que completa el mensaje se puede observar en *Content,* que contendrá siempre un String que corresponderá al String facilitado por el Json, con la información que teníamos dentro de un objeto Aircraft.

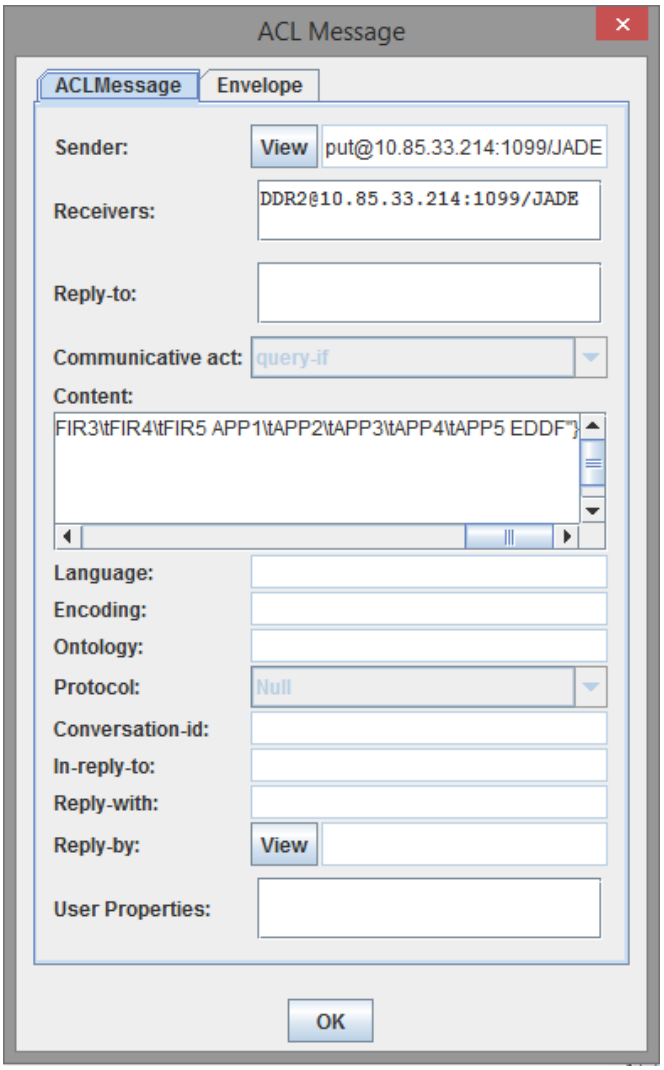


Figura 4-24: Información comprendida dentro de un mensaje ACL

En la figura 4-25 podemos ver una gráfica con todos los posibles mensajes entre agentes que se pueden dar en una simulación.

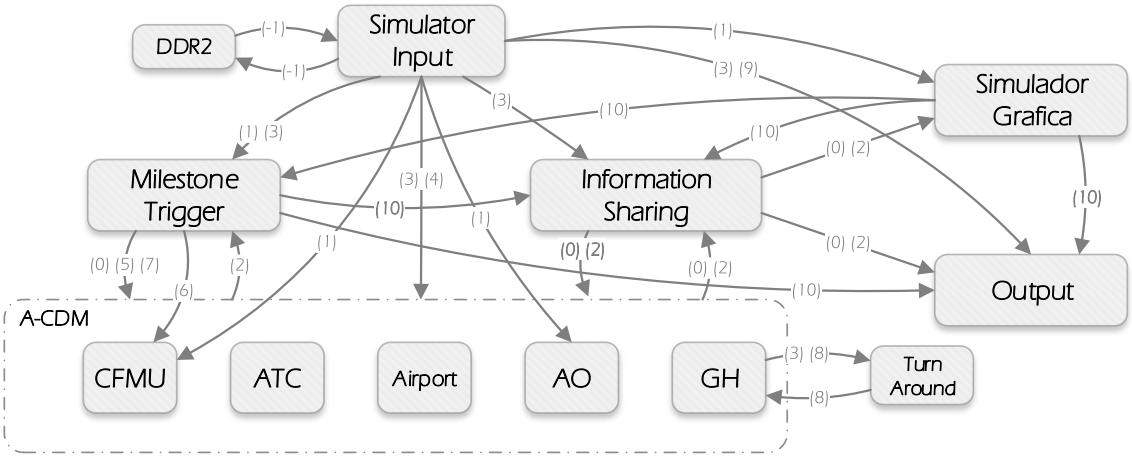


Figura 4-25: Comunicaciones entre agentes

Los números que observamos en la figura 4-25 indican el tipo de comunicación que se da entre los agentes en cada caso. Este valor se encuentra dentro del programa en la variable OpcionMensaje, dentro del objeto Aircraft que se manda en cada comunicación. Existen 12 posibles valores distintos de OpcionMensaje, i cada uno expresa un objetivo distinto:

- **OpcionMensaje = -1**: El objetivo es leer el documento DDR2, este mensaje se da entre el SimulatorInput i el agente DDR2. El primer mensaje es para solicitar la lectura, y a continuación se responderá para informar de si se ha leído correctamente.
- **OpcionMensaje = 0**: Avisan de que un avión ha superado un milestone. Cuando
- **OpcionMensaje = 1**: Mensajes avisando de que se puede realizar la lectura de los documentos de entrada que necesite cada uno de los agentes, también avisa de que ya se puede iniciar la simulación.
- **OpcionMensaje = 2**: Mensaje que avisa de un cambio en alguno de los tiempos del avión.
- **OpcionMensaje = 3**: Mensaje que avisa con los parámetros iniciales de simulación.
- **OpcionMensaje = 4**: Mensaje que manda el SimulatorInput a los agentes para avisar-les que durante la simulación tendrán de aplicar algún retraso
- **OpcionMensaje = 5**: Mensaje que manda el MilestoneTrigger a los agentes avisándoles de la hora actual para que apliquen los retrasos de tipo 1 cuando haga falta.
- **OpcionMensaje = 6**: Mensaje que mande el MilestoneTrigger al CFMU para informarle que en 3h tiene programada la salida un avión que solo realiza el despegue en nuestro aeropuerto.
- **OpcionMensaje = 7**: Mensaje que manda el MilestoneTrigger a los agentes avisándoles de la hora actual para que apliquen los retrasos de tipo 4 cuando haga falta.
- **OpcionMensaje = 8**: Mensaje de comunicación entre el agente GH i el agente TurnAround, en la que el GH pide el tiempo de turn around de un avión en concreto, i el agente TurnAround le contesta.
- **OpcionMensaje = 9**: Mensaje que se da entre el SimulatorInput i el agente Output para informarle de que tiene que mostrar por pantalla los resultados de un documento Results.txt de una simulación realizada anteriormente.
- **OpcionMensaje = 10**: Mensaje que envía el MilestoneTrigger o el SimuladorGráfica a los agentes InfoSharing, Output y MilestoneTrigger para avisarles de que se ha terminado la simulación, ya sea porque se han acabado los milestones, o porque el usuario ha escogido parar la simulación antes de tiempo.

4.4.2 ¿Qué se envía en un mensaje? Objeto AIRCRAFT

El programa se basa en la simulación del A-CDM en un aeropuerto, por lo que es necesario pensar en aviones llegando, saliendo y realizando las tareas comprendidas en el turn around dentro de nuestro aeropuerto. Por lo que es necesario crear un objeto con la información recopilada de cada uno de los aviones que actuarán en el aeropuerto de estudio. Este es el motivo por el que

se ha creado la clase Aircraft. Otro punto importante del CDM es que los agentes que forman el aeropuerto tienen la obligación de comunicarse entre ellos para informarse sobre las novedades de los aviones.

Puesto que los agentes informarían sobre los aviones, se decidió que lo más práctico era que cada vez que se mandara un mensaje entre agentes se mandara un objeto de la clase Aircraft que contendrá la nueva información sobre un avión que ha modificado alguno de sus parámetros.

Por estos motivos el objeto Aircraft estará preparado para informar sobre cualquier posible cambio en cualquier valor de un avión en concreto, pero también estará preparado para mandar una información incluso cuando en ese momento el objetivo no sea mandar la información de un avión en concreto.

La clase Aircraft contiene una serie de atributos y de métodos. Cada objeto Aircraft tiene los siguientes atributos:

- **id:** Contiene el Callsign del avión.
- **STATUS:** Indica el estado del avión:
  - INITIATED: Cuando se define el avión.
  - AIRBONE: Cuando acaba de despegar del aeropuerto de origen y se dirige al nuestro.
  - FIR: Cuando se encuentra dentro de la FIR de nuestro aeropuerto
  - FINAL: Una vez empieza con la fase de aproximación.
  - LANDED: Una vez ha aterrizado en el aeropuerto de estudio.
  - INBLOCK: una vez llega a la Gate.
  - SEQUENCED:
  - BOARDING: Está entrando el pasaje para el siguiente vuelo.
  - READY: Está preparado para salir de nuevo, espera órdenes.
  - OFFBLOCK: ha salido de la Gate y se dirige a la pista de despegue
  - DEPARTED: ha despegado de nuestro aeropuerto y se dirige a otro destino.
- **tiempos:** vector de 5 valores, cada posición del vector indica un tiempo diferente, [ETOT', ELDT, ETOT, EIBT, EOBT].
- **actualizadas**: vector de enteros, cada valor corresponde a un tiempo o dato que se ha actualizado en la última modificación del objeto. En la tabla 4-6 podemos observar la tabla de modificaciones con su valor correspondiente en el vector.

DATO ACTUALIZADO	VALOR
ID	1
STAUS	2
ETOT'	3
ELDT	4
ETOT	5
EIBT	6
EOBT	7
MILESTONE ACTUAL	8

AO	9
ADEP	10
ADES	11
MODELO	12
MATRICULA	13
ACGT	14
ASBT	15
ARDT	16
ASRT	17
TSAT	18
INCIDENCIAS	19
TFIR	20
TAPP	21
MILESTONE 9	22
MILESTONE 10	23

Tabla 4-6: Significado valores actualizadas

- **milestone:** Contiene el último milestone que ha superado el avión.
- **tiemposVuelo:** Vector de 2 posiciones, cada posición indica el valor de uno de los tiempo de vuelo, [TFIR, TAPP].
- **tiemposSecundarios**: Vector de 5 posiciones, cada posición indica el valor de un tiempo, [ACGT, ASBT, ARDT, ASRT, TSAT]
- **infoVuelo:** vector de 2 posiciones, caso uno indica un tiempo, estos valores son fijos, [Tiempo en FIR, Tiempo en APP]
- **nextMilestone**: booleano que indica si acabamos de superar un milestone o no.
- **opcionMensaje:** Sirve para definir la intención del mensaje, podemos ver en de la figura 4-25 los diferentes valores que puede tomar esta variable a lo largo de una simulación:
- **UltimoAgente:** Se guarda el nombre del último agente que modifico algún dato del vuelo.
- **ActuMilestone**: Contiene una lista de los milestones que se modifican. Cada posición del vector contendrá la siguiente información: [numeroMilestone nuevaHora] (un String separado por un espacio)
- **ADEP**: Aeropuerto de salida
- **ADES:** Aeropuerto de destino
- Modelo:
- **AO**: Aerolinea
- **Matricula:** Contendrá el Callsign2 cuando sea necesario
- **Categoría**: Categoría del avión. 1→Super Heavy; 2→ Heavy; 3→Medium; 4→Light
- **Booleano**: Contiene información sobre la respuesta de la última pregunta que se le ha hecho.
- **Incidencias:** String utilizado para guardar la información relacionada sobre todas las incidencias que tiene el avión a lo largo de la simulación. Cada variación en algún tiempo se guarda aquí. Las incidencias se guardan una detrás de la otra por orden de aplicación separadas por “/n” unas de las otras. El programa realizará filtrados a las incidencias para no repetirlas en caso de que dos agentes quieran

aplicar un mismo retraso. Cada incidencia contiene la siguiente información:
  - Tiempo Retrasado
  - Tiempo Antiguo
  - Tiempo Actualizado
  - Nombre Agente que añade el retraso
  - Hora del retraso
  - Comentarios (opcional)

En ciertas ocasiones se utiliza el apartado incidencias para enviar una información, una petición o un dato distinto a un avión. Por ejemplo cuando el agente SimulatorInput avisa al agente DDR2 de que empiece con la lectura del documento DDR2, en el apartado incidencias se puede encontrar la ruta del documento DDR2 entre otros valores.

El conjunto de estos valores es el que forma y da sentido al objeto Aircraft, pero con tal de poder aprovechar al máximo este objeto necesita una serie de métodos:

- **Gets**: permiten solicitar información de un parámetro en concreto de la clase.
- **Sets**: permiten modificar o guardar información de un parámetro en concreto de la clase
- **AddActualizadas**: Permite añadir una serie de actualizadas, sin eliminar las ya existentes.
- **añadirIncidencia**: añade una Incidencia a la lista de las incidencias que ya tenía previamente.
- **añadirIncidenciaCompleta**: añade una Incidencia a la lista de incidencias que ya tenía previamente sin hacer comprobaciones de si esa incidencia ya se había escrito previamente. El objetivo de esta función es permitir a los distintos agentes poder escribir una incidencia con el formato deseado (distinto al formato estándar de las incidencias que se añadirán para cada retraso que sufre un avión). De esta manera los agentes pueden mandar mensajes con la información deseada a los otros agentes cuando sea necesario.
- **newStatus:** Función para cambiar el valor de STATUS, se le entra un numero entero y en función del valor el STATUS se modifica según la tabla 4-7:

STATUS	VALOR
INITIATED	0
AIRBONE	1
FIR	2
FINAL	3
LANDED	4
INBLOCK	5
SEQUENCED	6
BOARDING	7
READY	8
OFFBLOCK	9
DEPARTED	10

Tabla 4-7: Valores de STATUS en Aircraft

- **printFlight:** Se usa para mostrar en pantalla la información guardada en el objeto, en función del número con el que se llama a la función se mostrará una información u otra.

El objeto Aircraft también tiene dos constructores tal y como podemos observar en la figura 4-26:

```
// Constructors

//Primer constructor
public Aircraft()
{
    super();
    this.nextMilestone = false;
}

//Segundo constructor
public Aircraft(String id, String sTATUS, int[] tiempos)
{
    super();
    this.id = id;
    STATUS = sTATUS;
    this.tiempos = tiempos;
    this.nextMilestone = false;

}
```

Figura 4-26: código constructores objeto Aircraft

El primer constructor sirve para inicializar el objeto Aircraft vacío, el segundo en cambio permite inicializarlo dándole una serie de valores iniciales como el id, el STATUS y los tiempos, de manera que se pueda inicializar el objeto de manera más eficaz si ya sabemos esta serie de datos.
Hasta aquí todo el código situado dentro del objeto Aircraft. Este objeto es utilizado en todos los agentes, ya sea para guardar información de cualquier avión o porque se requiere mandar un mensaje y se decide utilizar esta clase.

4.4.3 ¿Dónde se guardan los aviones? Objeto TABLA

En los agentes cada avión de la simulación está guardado en un objeto de tipo Aircraft, eso significa que llegaremos a tener grandes cantidades de objetos Aircraft en simulaciones de un día completo cuando se trate de aeropuertos grandes, es por esto que se ha decidido que estos agentes guardaran la información en una tabla de aviones. Por este motivo se ha creado la clase tabla, la cual está preparada para facilitar al programa el guardar, extraer y trabajar con los aviones.

En el momento de crear el objeto tabla se necesita saber la máxima cantidad posible de aviones que llegaremos a tener. Por eso se inicializa con el constructor de la figura 4-27:

```
public Tabla (int n)
{
    aviones = new Aircraft[n];
    numero = 0;
}
```

Figura 4-27: Código Constructor Objeto Tabla

Donde *n* corresponde al número de aviones que se han leído del documento input.txt escrito por el agente DDR2, pues esos son todos los aviones que simularemos.
La clase tabla está formada de dos atributos:

- **numero**: Valor numérico que indica el número de aviones guardados dentro de la tabla.
- **Aviones**: Vector de longitud indefinida, en cada posición guarda un objeto del tipo Aircraft. Una posición por cada avión del que se tiene información.

Como hemos dicho el único objetivo de la clase tabla no es guardar los aviones, sino ayudarnos a trabajar mejor con ellos, es por esto que se han definido los siguientes métodos:

- **getAviones**: devuelve la tabla entera con todos los aviones.
- **setAviones:** Guarda una tabla entera de aviones.
- **getnumero**: devuelve el valor de *numero*
- **setnumero**: añades el valor de *numero*
- **setAvion**: añade un avión a la tabla o actualiza un avión ya existente en la tabla.
- **getAvion**: devuelve un avión de la tabla
- **printTabla**: muestra por pantalla la información relacionada con un avión de la tabla
- **PosicionTabla**: mira si un avión existe dentro de la tabla, y en qué posición se encuentra. En caso de no existir devuelve la posición en la que se encontraría si se añadiese.
- **GetPositionAvion**: a partir de un Callsign mira si existe en la tabla algún avión con el mismo Callsign y devuelve el índice. Si en la tabla no tenemos ningún avión con dicho Callsign devuelve -1
- **contarAviones**: cuenta el número total de aviones en la tabla y lo guarda en la variable *numero*.

Mediante esta serie de métodos, podemos guardar y trabajar de forma óptima con todos los aviones sin muchas complicaciones. Este objeto es utilizado constantemente por los agentes CDM, el InfoSharing, el Output y algunas interfaces también han hecho uso de ellas para mostrar por pantalla los datos de la tabla de forma optimizada

4.5 *Documentos de entrada y Resultados*

4.5.1 Documentos INPUT

Con tal de hacer funcionar correctamente el simulador, este requiere de una serie de datos de entrada. A continuación se explicara cuáles son los documentos de entrada que espera recibir el programa, y que información contiene cada uno. Se esperan 4 documentos de entrada:

- Documento de tráfico aéreo DDR2: es el documento más importante, es el encargado de añadir el tráfico aéreo de un día al simulador, se obtiene de la base de datos oficial de Eurocontrol, llamada DDR2 (Demand Data Repository). Estos datos se encuentran gratis para investigadores y estudiantes a través de la página web de Eurocontrol. Al descargar este documento vendrá en formato .txt o .SO6.

Este documento contiene toda la información de los vuelos que se tendrán en cuenta para la simulación. En el apéndice de "Obtención Datos DDR2 de Eurocontrol", podemos encontrar una guía detallada sobre cómo obtener este documento en el formato que lee el programa (documento DDR2).

Si abrimos el documento podremos observar que este contiene mucha información. Cada línea da información sobre el recorrido de un avión en cada uno de los segmentos en el que se puede fraccionar el vuelo completo. Esta información esta almacenada tal y como vemos en la tabla 4-8:

#	Información	Tipo	Tamaño	Comentarios
1	Identificador de segmento	char		Inicio segmento "_" final segmento
2	Inicio del vuelo	char	4	Código ICAO
3	Final del vuelo	char	4	Código ICAO
4	Modelo avión	char	4	
5	Hora inicio segmento	num	6	HHMMSS
6	Hora final segmento	num	6	HHMMSS
7	FL inicio segmento	num	1-3	
8	FL final segmento	num	1-3	
9	Estado	num	1	0=subiendo, 1 bajando, 2 crucero
10	Callsign	char		
11	Data inicio segmento	num	6	YYMMDD
12	Data final segmento	num	6	YYMMDD
13	Latitud inicio segmento	num		Minutos y decimales
14	Longitud inicio segmento	num		Minutos y decimales
15	Latitud final segmento	num		Minutos y decimales
16	Longitud final segmento	num		Minutos y decimales
17	Identificador de vuelo	num		Numero único para cada vuelo
18	Secuencia	num		Empieza en 1 y aumenta a cada segmento
19	Longitud del segmento	num		En millas náuticas
20	Color del segmento	num		

Tabla 4-8: Información comprendida en cada línea del documento DDR2

Cada línea del documento extraído de Eurocontrol contiene la sobre un segmento de vuelo de un avión en concreto, Las líneas consecutivas pertenecen a segmentos consecutivos de un mismo avión, por lo que un avión va añadiendo valor a la secuencia 1, 2, 3, 4, etc. hasta que se terminen los segmentos del vuelo de ese avión, momento en el que el documento empieza con la información del siguiente avión

- Documento de parámetros del aeropuerto: El documento airport.txt es también uno de los documentos importantes. Este documento

contendrá toda la información que afectara a las características de nuestro aeropuerto. Es un documento que será creado manualmente por el usuario antes de empezar la simulación, con tal de poder definir los parámetros que se deseen.

El documento estará compuesto por 8 líneas, cada una de las cuales dará una información relacionada con el aeropuerto.

Siguiendo el esquema de la figura 4-9 se escribirá todo el documento Airport.txt, la información de este documento se usara en el programa para definir los parámetros iniciales.

línea	Contenido	Ejemplo
Línea 1	Código ICAO del aeropuerto a estudiar	LEBL
Línea 2	Aerolíneas con agente propio	RYR IBE VLG
Línea 3	Operaciones/h de llegada	Op/h-In 25
Línea 4	Operaciones/h de salida	Op/h-Out 30
Línea 5	Categorías de los aviones	A380-1 B744-2 A346-2 B773-2
Línea 6	Waypoints de entrada a la FIR	PERDU ANETO GIROM PUMAL
Línea 7	Waypoints de inicio de la aproximación	GRAUS LOBAR CASPE SENIA
Línea 8	Contratos de los GH	IBEServices-IBE Lesma-RYR

Tabla 4-9: Ejemplo Información documento Airport

Es importante que en la *Run Configurations* del programa queden bien definidos las aerolíneas y los agentes GH con agente propio que aparecen en este documento para que el programa pueda funcionar correctamente, mirar apéndice “Manual del Software”.

- Documento de retrasos: El documento de retrasos es un documento de entrada no obligatorio, podemos decidir que no queremos aplicar retrasos en una simulación, en cuyo caso no será necesario disponer de este documento. También podría ser que queramos aplicar retrasos en la simulación, pero estos no los cojamos de un documento externo, sino que podemos usar las interfaces del programa para crear los retrasos a nuestra medida. De esta manera crearíamos un nuevo documento de Retrasos.txt.

Por último el programa permite leer un documento externo de retrasos.txt, con tal de aplicar unos retrasos determinados, podría darse el caso de que deseáramos usar unos retrasos en concreto para comparar 2 simulaciones o que nos dieran un documento de retrasos y nos pidieran una simulación con él. En cualquier caso el programa permitiría usar este

documento integro, o modificarlo para añadirle algún otro retraso a los que aparezcan de antemano en el documento, o eliminar alguno de ellos.
Podemos ver cómo estará escrito el documento de retrasos en el apartado 4.6.1.1

- Turn-Around-Time.txt: El documento Turn-Around-Time.txt contiene una lista con el tiempo mínimo necesario para realizar el turn around cada avión en función del modelo y la aerolínea. Este documento no es obligatorio. En caso de no existir, o estar escrito en un formato no correcto, el simulador supondrá que todos los aviones tardarán un tiempo de 35 min en hacer el turn around en caso general, aumentando este valor con los aviones de categorías más pequeñas (aviones grandes). Este documento será siempre leído por el agente TurnAround. La información está estructurada de manera que a cada línea encontramos el tiempo de turn around para un modelo de avión y una aerolínea, la aerolínea no estará siempre identificada, en esos casos se considerará que la información es válida para todas las aerolíneas
Ejemplo:

```
A320 35
A320 37     VLG
B767 55
GEN  37
```

En el ejemplo un A320 de la aerolínea Vueling (VLG) tendría un tiempo de turn around de 37 minutos como mínimo, mientras que cualquier otro A320 que no fuera de Vueling tendría un tiempo de turn around de 35 min. Se puede incluir una variable GEN, esta indicara el tiempo mínimo para el turn round a los aviones cuyos modelos no aparezcan en la lista. El valor de GEN siempre será mayor o igual a 35, en caso de no aparecer se considerara de 35min el tiempo mínimo para los aviones de categoría 4.

4.5.2 Otros documentos input (Simulator Input)

Estos son los 4 documentos de entrada para iniciar el programa, pero los agentes Simulator Input y DDR2, mediante la información de Airport.txt y del documento de Eurocontrol, creará una serie de documentos que necesitaran otros agentes posteriormente.
Los documentos que se crearán son:

- Input.txt (DDR2): El agente DDR2 es el encargado de crear el documento Input.txt mediante la información extraída del documento de Eurocontrol y del Airport.txt tal y como se ha explicado al definir el agente DDR2.

El documento Input.txt contiene una línea de texto para cada aterrizaje y para cada despegue que suceda en el aeropuerto estudiado.
Ejemplo:

```
1    DLH1TR    B753 LEPA EDDF 0925 1117 1122 1127
2    DLH1TR    A320 EDDF LDZA 1049 DLH1EM
```

El documento Input.txt será el que leerá el SimulatorInput para escanear toda la información y poder escribir otros documentos que utilizarán los otros agentes a la hora de simular todo el proceso.

- FlightPlans.txt (SimulatorInput): Este documento es escrito por el SimulatorInput después de leer el documento input.txt, en el reescribe la información relacionada con los aterrizajes a nuestro aeropuerto de una manera que sea más fácil de leer para el agente CFMU. Contendrá una línea con la información de la figura 4-10 para cada vuelo que llega al aeropuerto.

Callsign	Aerolínea	Modelo	Aeropuerto OUT	Aeropuerto IN	ETOT'	TFIR	TAPP	ELDT
IBE450	IBE	A321	LEPA	LEBL	0910	0946	0951	0956

Tabla 4-10: Formato escritura documento FlightPlans.txt

- FlightPlansSalidas.txt (SimulatorInput): Este documento es escrito por el SimulatorInput después de leer el documento input.txt, en el el SimulatorInput escribe la información relacionada con los despegues de nuestro aeropuerto de manera que sea más fácil de leer para el agente CFMU. Cada línea contendrá la información de la tabla 4-11 para cada vuelo:

Callsign	Aerolínea	Modelo	Aeropuerto OUT	Aeropuerto IN	ETOT
RYR58HA	RYR	B736	LEBL	LEIB	1325

Tabla 4-11: Formato escritura FlightPlansSalidas.txt

- TriggerCDM.txt (SimulatorInput): El documento TriggerCDM.txt es el documento que contendrá la información sobre los primeros milestones de cada vuelo y sobre los retrasos que afectan a múltiples aviones. Este documento será leído por el agente MilestoneTrigger, para iniciar el proceso de informar a todo el mundo de que se ha producido un milestone, y se puedan realizar las acciones correspondientes en cada caso. El documento contendrá la siguiente información:

- **Llegadas**: Para todos los aviones que llegan al aeropuerto de estudio, se guardarán en el documento la información correspondiente a los 6 primeros milestones. Sin importar si se quedaran en nuestro aeropuerto. En este caso se escribirá en el documento:

`Callsign 1 HHMM` → Donde HHMM equivale a ETOT'-3h
`Callsign 2 HHMM` → Donde HHMM equivale a ETOT'-2h
`Callsign 3 HHMM` → Donde HHMM equivale a ETOT'
`Callsign 4 HHMM` → Donde HHMM equivale a TFIR
`Callsign 5 HHMM` → Donde HHMM equivale a TAPP
`Callsign 6 HHMM` → Donde HHMM equivale a ELDT

- **Salidas**: Para los aviones que solo salen se guardara información con un milestone inventado "-1", 3 horas antes del

ETOT, que representará el momento en el que entra el avión en la simulación y se empieza a considerar sus tiempos para en la simulación, hasta este momento no se ha realizará ningún cálculo de tiempos para estos aviones. Se han escogido 3 horas antes de la ETOT, porque a 3 horas del despegue de un avión este supera el milestone 1 para el aeropuerto de destino. En este caso se escribirá en el documento:

`Callsign -1 HHMM` → Donde HHMM equivale a ETOT-3h

- **Avisos Hora (Retrasos de múltiples aviones)**: En ciertas situaciones se necesita que el programa mande un mensaje avisando simplemente de la hora de simulación en la que nos encontramos. Esto sucede cuando tenemos retrasos de tipo 1 o tipo 4. En este caso se escribirá en el documento:

`Retraso 0 HHMM` → Retraso tipo 1
`RetrasoPista 0 HHMM` → Retraso tipo 4

- AO1.txt (SimulatorInput): Documento reducido con información de todas las llegadas a nuestro aeropuerto. Este documento se escribe como paso medio para la creación del documento VuelosAerolineas.txt. Además puede ser leído por el usuario fácilmente por tener la información muy simplificada, así puede ver la información relacionada con todas las llegadas. Cada línea corresponde a una llegada y se escribe de la siguiente manera:

Callsign – Aerolínea – Modelo - ELDT
EZY87ER EZY A319 1144

- AO2.txt (SimulatorInput): Documento reducido con información de todas las salidas desde nuestro aeropuerto. Este documento se escribe como paso medio para la creación del documento VuelosAerolineas.txt. Además puede ser leído por el usuario fácilmente por tener la información muy simplificada, así puede ver la información relacionada con todas las salidas. Cada línea corresponde a una salida y se escribe de la siguiente manera:

Callsign – Aerolínea – Modelo – ETOT - Callsign2
EZY87ER EZY A319 1500 EZY80ER

- VuelosAerolineas.txt (SimulatorInput): Documento escrito mediante AO1.txt y AO2.txt, contiene información muy breve sobre todas las llegadas y salidas del aeropuerto de estudio. Este documento será usado por el agente AO. El agente AO usara el documento VuelosAerolineas.txt para relacionar los FlightPlans de las llegadas y las salidas, con la información del documento podrá ver si cualquier avión que llegue al aeropuerto lo hace para quedarse o tiene preparada la salida en otro momento. El documento está escrito de la siguiente manera. Cada línea

corresponde a un avión sin importar si solo llega, si solo sale o si realiza las dos acciones.

Callsign - Aerolínea – Modelo – ELDT – ETOT

EZY87ER	EZY	A319	1144	1500
NAX5175	NAX	B738	2129	9999
RYR58HA	RYR	B738	9999	435

Si nos fijamos en los valores ELDT y ETOT, podemos observar que en algunos casos marca un “9999” porque realmente no realiza esa acción en ningún momento dentro de la simulación. Eso significa que ese avión solo llega o solo sale del aeropuerto.

ELDT = 9999 → Avión que solo despega, no aterriza

ETOT = 9999 → Avión que solo aterriza, no despega

4.5.3 Documentos Output:

El simulador facilita unos datos de salida, ya sea por pantalla mediante las interfaces o mediante los conocidos como documentos output para entregar al usuario los resultados de la simulación. A continuación en este apartado se explicarán los diferentes documentos de salida que encontraremos:

- Results.txt: es el documento que contendrá la información de todos y cada uno de los aviones simulados. Tendrá una línea de información para cada avión. En cada caso la información escrita en el documente tendrá la información de todos los aviones simulados, escribiendo a cada línea del documento el objeto Aircraft de un avión siguiendo un formato String escrito por el Json (de la misma manera que se hace al enviar un mensaje entre agentes).
Aunque este documento está pensado como documento de salida, el programa está preparado para leerlo y mostrar la información por pantalla, aunque no aplicará cálculos ni variaciones a estos datos, el programa podrá leer este documento para exponer los resultados en una interfaz facilitando la lectura y el entendimiento de los datos. Este documento será escrito por el agente Output al recibir un mensaje avisando de que se ha terminado la simulación.
Este documento también podrá ser leído después de realizar otra simulación con el objetivo de poder comparar y analizar los resultados de 2 simulaciones distintas a la vez.

- MilestonesResultados.txt: Como se ha explicado el agente MilestoneTrigger está mandando continuamente información de nuevos milestones con tal de mantener el programa en funcionamiento. Con el objetivo de poder hacer un seguimiento de los milestones que se han realizado durante la simulación, este agente escribirá cada vez que mande un mensaje, una línea en el documento MilestonesResultado.txt, donde la línea contendrá la información de la que quería avisar en cada caso el MilestoneTrigger.

- ParametrosSimulación.txt: Documento que contiene los parámetros utilizados durante la simulación, de manera que si el usuario quiere repetir una simulación, puede mirar este documento y ya sabrá cuales eran los parámetros con los que se realizó la primera simulación. Este documento está escrito por la interfaz del agente SimulatorInput justo antes de empezar con la simulación. La información que encontraremos en este documento es la siguiente:

```
TaxiIntime  15
TaxiOutTime 20
VelocidadMilestones   250
RutaInput   C:/Users/xavi/workspace/SimuladorV1.6/
Aviones 603
LEBL
RYR     IBE  VLG  AEA  EZY  DLH  NAX  AFR
Op/h-In 25
Op/h-Out    30
A380-1  B744-2  A346-2  B773-2  A343-2  B772-2  A333-2
A332-2
FIR1    FIR2 FIR3 FIR4 FIR5
APP1    APP2 APP3 APP4 APP5
IBEServices-IBE  Lesma-RYR FrankfurtAGS-TAP-DUB-VLG
```

Donde cada línea informa sobre un parámetro utilizado en la simulación. Si nos fijamos, la información encontrada entre el sexto parámetro (incluido) hasta el último, es exactamente la información que teníamos en el documento Airport.txt de entrada al simulador, de manera que si conservamos este documento, pero perdemos el del Airport.txt, es muy fácil recuperarlo.

4.6 *Funcionalidades Extra*

4.6.1 Retrasos en la simulación

En la vida real los aviones sufren retrasos de todo tipo, averías, meteorología adversa, viento, los pasajeros entran lento al avión… Por ello queríamos que el programa fuera capaz de aplicar cualquiera de estos retrasos siempre que así lo deseará el usuario.

Para esta tarea primero se ha pensado en que tipos de retrasos queremos que pueda aplicar el simulador, y que información necesitará el programa para realizar la tarea. Para eso se decidió permitir al usuario entrar 3 tipos de retraso, más uno extra que afectará a la capacidad de las pistas y nos centraremos en el en el apartado 4.6.2

- **Retrasos de tipo 1**: Afectará a todos los aviones en un tiempo determinado

- **Retrasos de tipo 2**: Afectará a un avión y se conoce el motivo, por lo que se sabe quién aplica el retraso, que tiempo afecta…
- **Retrasos de tipo 3**: Afectará a un avión y no se conoce el motivo, por lo que necesitaremos información sobre quien aplicará el retraso, a que tiempo afectará, cuando se aplica el retraso…

También explicaremos como realiza el programa para aplicar estos retrasos en la simulación. Explicaremos como vendrán definidos estos retrasos en el documento externo (creado por el usuario mediante interfaz), como leerá el programa este documento y como actuará el simulador para aplicar todos los retrasos en el momento correspondiente.

4.6.1.1 Documento de retrasos (Documento + Interfaz)

El programa leerá los retrasos de los aviones del documento retrasos.txt, este documente contendrá la información de los retrasos a aplicar, pero se puede escribir o modificar la información ya existente en el documento utilizando las interfaces gráficas que proporciona el programa antes de iniciar la simulación.

Los retrasos estarán escritos en el documento siguiendo la estructura que encontraremos a continuación, para que el programa pueda entender los retrasos y aplicarlos correctamente, podemos tener 4 tipos de retrasos distintos:

- **Retraso Tipo 1**: Afecta a todos los aviones que durante un periodo de tiempo determinado tienen se encuentran realizan un milestone determinado. Se puede filtrar los aviones afectados según las aerolíneas y modelos de avión.
- **Retraso Tipo 2**: Afecta a un solo avión, en un momento determinado, se conoce el motivo del retraso. Debido a que se conocen las causas del retraso, se puede saber quién es el encargado de aplicar el retraso, y cuál es el tiempo que se retrasará.
- **Retraso Tipo 3**: Afecta a un solo avión, en un momento determinado, no se conoce el motivo del retraso, pero se tiene información sobre en qué tiempo se aplica el retraso, y cuál es el agente que aplica lo aplica.
- **Retraso Tipo 4**: El último tipo de retraso indica una reducción de la capacidad de pista para las llegadas o para las salidas en un periodo de tiempo determinado. Por lo que al tener este tipo de retraso, las distancias entre dos despegues o dos aterrizajes se verán incrementadas.

Se han escogido estos 4 tipos de retrasos, pues nos permiten simular casi cualquier tipo de problemas que pueda tener un aeropuerto. Desde problemas meteorológicos o problemas en taxi way con los retrasos tipo 1, retrasando a todos los aviones. Podemos decidir causar problemas en un solo avión en un momento determinado permitiendo simular casi cualquier escenario deseado con el tipo 2 y el tipo 3.
Mediante el tipo 4 podemos simular problemas meteorológicos en pista, o algún otro problema que pueda afectar a la capacidad de pista de aterrizaje o despegue.

A continuación podemos observar cómo se escribirán cada uno de los retrasos en el documento retrasos.txt:

Retraso tipo 1:
Se escribirá con el siguiente código:

```
1 Hinicio Hfinal Hconocimiento Retraso Situación MilestoneAfectado AOAfectadas ModelosAfectados Comentarios
```

- El primer valor es un 1, para informar que nos encontramos con un retraso de tipo 1.
- Hinicio: Indica la hora a la que los aviones empezaran a sufrir los retrasos.
- Hfinal: indica la hora a la que finalizan los problemas, pasada esta hora los aviones ya no sufrirán retrasos.
- Hconocimiento: hora en la que se conoce sobre este retraso, es la hora en la que el simulador podrá tener en cuenta que aviones sufrirán retrasos, calcularlos, y también calcular todos los efectos que estos tendrán en el aeropuerto.
- Retraso: cantidad de minutos que tendremos de retrasar a los aviones afectados.
- Situación: Valor entero que indicara la situación por la que se da el retraso según el código de retrasos de la tabla 4-12, en la tabla también podemos observar que tiempo se retrasará y quien aplicará el retraso.
- MilestoneAfectado: Miraremos si dentro del periodo de tiempo descrito entre la Hinicio y la Hfinal, algún avión se encuentra realizando este milestone, es caso afirmativo este avión será afectado por el retraso.
- AOAfectadas: Lista de Aerolíneas que se verán afectadas por el retraso. Cada aerolínea vendrá separada con un "-"de las otras aerolíneas afectadas. Ejemplo: VLG-RYR-IBE, de esta manera el retraso solo no será aplicado a un avión cuya aerolínea no aparezca en esta línea. Si queremos que afecte a todos los aviones sin tener en cuenta su aerolínea escribiremos "-"
- ModelosAfectados: Lista de modelos de avión que se verán afectados por el retraso. Cada modelo vendrá separada con un "-"de los otras modelos afectadas. Ejemplo: A320-A340-B747-A380, de esta manera el retraso solo no será aplicado a un avión cuya modelo no aparezca en esta línea. Si queremos que afecte a todos los aviones sin tener en cuenta su modelo escribiremos "-"
- Comentarios.

Ejemplo retraso tipo 1:

```
1 1000 1100 0900 25 2 15 VLG-RYR A320-A340-B767 comentarios
```

Retraso tipo 2:
Se escribirá con el siguiente código:

```
2 Callsign MilestoneConocimiento Retraso Situación Comentarios
```

- El primer valor es un 2, para informar que nos encontramos con un retraso de tipo 2.
- Callsign: Indica el callsign del avión al que se le aplicara el retraso.
- MilestoneConocimiento: indica el milestone en el que se encuentra el avión en el momento que se sabe que tendrá de aplicar el retraso en cuestión.
- Retraso: cantidad de minutos que tendremos de retrasar al avión.
- Situación: Valor entero que indicara la situación por la que se da el retraso según el código de retrasos de la tabla 4-12. También indicara que tiempo se retrasará y quien aplicará el retraso.
- Comentarios

Ejemplo retraso tipo 2:

```
2 RYR300 3 10 2 Comentarios
```

Retraso tipo 3:
Se escribirá con el siguiente código:

```
3 Callsign MilestoneConociemiento Retraso TiempoRetrasado QuienRetrasa Comentarios
```

- El primer valor es un 3, para informar que nos encontramos con un retraso de tipo 3.
- Callsign: Indica el callsign del avión al que se le aplicara el retraso.
- MilestoneConocimiento: indica el milestone en el que se encuentra el avión en el momento que se sabe que tendrá de aplicar el retraso en cuestión.
- Retraso: cantidad de minutos que tendremos de retrasar al avión.
- TiempoRetrasado: Indica cual es el tiempo donde el avión recibirá el retraso. Se pueden poner todos los tiempos desde la TFIR hasta el ETOT. Siempre teniendo en cuenta que si en el momento del MilestoneConocimiento el avión ya ha superado el TiempoRetrasado, este no se retrasará, pues no se puede retrasar un tiempo que ya ha sucedido.
- QuienRetrasa: Indica cual es el agente encargado de aplicar el retraso al avión.
- Comentarios

Ejemplo retraso tipo 3:

```
3 RYR300 3 15 ELDT ATC Comentarios
```

Retraso tipo 4:
El tipo 4 indica siempre reducción de la capacidad de la pista de aterrizajes, a la pista de despegues o ambas.

Se escribirá con el siguiente código:

```
4 Hinicio Hfinal Hconocimiento NuevasCapacidades(DEP/ARR)
Comentarios
```

- El primer valor es un 4, para informar que nos encontramos con un retraso de tipo 4.
- Hinicio: Indica la hora a la que los aviones empezaran a sufrir los retrasos.
- Hfinal: indica la hora a la que finalizan los problemas, pasada esta hora los aviones ya no sufrirán retrasos.
- Hconocimiento: hora en la que se conoce sobre este retraso, es la hora en la que el simulador podrá tener en cuenta que aviones sufrirán retrasos, calcularlos, y también calcular todos los efectos que estos tendrán en el aeropuerto.
- NuevasCapacidades: dos valores separados por un espacio, el primero indica el número de Take-Off por hora que tendremos con la nueva capacidad de pista, y el Segundo indica el número de llegadas por hora que tendremos con la nueva capacidad de pista. Si solo queremos aplicar reducción en una de las dos pistas, pondremos un valor de 30 o superior a la pista que no es afectada por este retraso.
- Comentarios

Ejemplo retraso tipo 4:

```
4 1000 1100 0900 10 30 Comentarios
```

Situación

En los retrasos tipo 1 y 2, tenemos la variable “Situacion”, esta variable se usa para indicar las situaciones de los retrasos tipo 1 y 2, a la vez que nos da información sobre qué agente aplicara el retraso y cuál será el tiempo retrasado. El valor de “Situación” será un entero que indicara según el código de la tabla 4-12 uno de los siguientes problemas

Situación	*Motivo*	*Quien lo aplica*	*Tiempo que retrasa*
0	Condiciones climáticas adversas en la FIR	CFMU	TAPP
1	Problemas en el Taxi Way de entrada al aeropuerto	Aeropuerto	ACGT
2	Problemas en el Taxi Way de salida del aeropuerto	Aeropuerto	ETOT
3	Retraso en el GH	GH	EOBT

Tabla 4-12: Tabla de Situaciones causantes de retrasos

Si el usuario quisiera ampliar esta lista de posibles motivos de retraso, se podría modificando un poco el código. Únicamente tendría que entrar en el SimulatorInput, a la función Getreceptor, y en el apartado “Efectos de X”, escribir para X=4, X=5... Quien será el encargado de aplicar cada uno de estos nuevos retrasos.

Tener en cuenta que si se desea que el receptor sea el GH, se tiene que copiar el código del caso X=3, pues tiene en cuenta que existen más de un agente GH
Si se quiere indicar que el agente encargado de aplicar el retraso sea el AO, también se tendrá de tener en cuenta que existen más de un agente AO, por lo que se tendrá de avisar al AO correspondiente al avión retrasado si se trata de un retraso tipo 2. Si se trata de un retraso tipo 1 se tendrá de avisar a todos los agentes AO.

También tendremos de indicar para cada nuevo valor posible de Situación que tiempo será el que se retrasará. Para dar esta información al programa entraremos en el agente CDM, y en la función TempsQueRetrasem pondremos el tiempo a retrasar en cada nuevo valor de X.

4.6.1.2 Ejecución de los retrasos

Anteriormente se ha explicado cómo escribir el documento de retrasos.txt, con el objetivo de poder aplicar distintos retrasos a los aviones, o de poder aplicar una reducción en la capacidad de la pista.
Si nos centramos en explicar donde se aplican estos retrasos dentro del código, no tenemos una parte concreta que se centre en ellos, pues para los retrasos intervienen muchos agentes:

- SimulatorInput
- MilestoneTrigger
- Airport
- AO
- ATC
- CFMU
- GH

SimulatorInput
El primer implicado es el SimulatorInput. Este agente será el encargado de leer el documento de retrasos.txt y empezar a tratar con la información que este contiene, con tal de avisar a los agentes implicados para cada retraso distinto de la lista.

Para que el simulador tenga en consideración el documento retrasos.txt, y por consiguiente se apliquen los retrasos, el usuario tendrá de realizar una serie de acciones previas al inicio de la simulación. Primero tendrá de *administrar retrasos*, desde la interfaz gráfica de *InterfazFormularioRetrasos*. Una vez allí podrá crear nuevos retrasos y decidir si tener en cuenta los retrasos ya escritos en algún documento. Al terminar de definir los retrasos, tendrá de avisar al programa de que esos son los que quiere aplicar mediante el botón *aceptar* de la *InterfazFormularioRetrasos.* En ese caso el programa escribirá en el documento retrasos.txt todos los retrasos que se tendrán en cuenta, vengan de otro documento o hayan sido creados por el usuario. Y estos serán los retrasos con los que el simulador trabajará.

Si el usuario no entrase en la interfazFormularioRetrasos1, por mucho que exista un documento retrasos.txt, el programa no los tendrá en cuenta para esta simulación.

Primero leerá el documento retrasos.txt en la función leerInputretrasos, y se guardara la información de cada línea del documento, acto seguido, mediante la

función *MandarRetrasos*, procederemos a analizar uno por uno los retrasos, y a avisar a los agentes implicados en la aplicación de cada uno de los retrasos. Como siempre que se manda un mensaje entre agentes, se mandara la información en una clase Aircraft, por lo que la información que se mandara para los retrasos contendrá siempre la información del retraso en las incidencias del avión.

Con tal de que los agentes que reciben la información de los retrasos sepan de qué se trata, la OpciónMensaje contendrá un 4, que es el indicador de que se trata de una información relacionada con los retrasos de los aviones.

Como se ha dicho anteriormente, el mensaje lo recibirán solo los agentes interesados en cada uno de los retrasos, por lo que se entrara en la función GetReceptor para cada retraso. En esta función realizaremos un primer escaneado dentro de la información relativa al retraso, con el objetivo de saber quién necesitará recibir el mensaje, para poder aplicar el retraso en cuestión en un futuro.

Una vez estamos dentro de la función *GetReceptor*, lo primero que miraremos es qué modo de retraso tenemos. En los retrasos de tipo 1 y tipo 2, el agente que se tendrá de informar del retraso dependerá del valor que tenga el Situación según la tabla 4-13, pues en cada caso será un agente distinto el que realizará la tarea de aplicar el retraso correspondiente.

Situacion	*Quien lo aplica*
0	CFMU
1	Aeropuerto
2	Aeropuerto
3	GH

Tabla 4-13: Agentes encargados de aplicar un retraso

Si tenemos un retraso de tipo 3, el agente al que informar viene dado en la misma línea con la información del retraso en la sexta posición:

Retraso tipo 3:

```
3 Callsign MilestoneConocimiento Retraso TiempoRetrasado
QuienRetrasa Comentarios
```

Los retrasos de tipo 4 se explican el en apartado de Reducción de capacidad de pista de manera detallada (apartado 4.6.2).

Una vez conocemos quién es el encargado de aplicar cada retraso se le mandará un mensaje. Se mandara un mensaje al agente por cada retraso en el que intervenga.

El SimulatorInput realizará una última función para poder procesar correctamente los retrasos de tipo 1 y 4. Añadirá en la lista de Milestones del documento TriggerCDM.txt una línea para cada retraso de tipo 1 y tipo 4, indicando la Hconocimiento, con el objetivo de que durante la simulación al llegar

a dicha hora, el MilestoneTrigger pueda mandar un mensaje a los agentes avisando de que tienen de aplicar los retrasos.

Como el documento TriggerCDM.txt contiene todos los milestones ordenados según la hora a la que sucederán, el programa añadirá la línea de avisar de la hora, en la línea correspondiente del documento para la Hconocimiento.
En el caso de tratar de un retraso de tipo 1 la línea a añadir será:

```
"Retraso  0    Hconocimiento"
```

En el caso de un retraso de tipo 4 la línea a añadir será:

```
"Retrasopista  0    Hconocimiento"
```

MilestoneTrigger:

El MilestoneTriger solo interviene en los retrasos de tipo 1 y tipo 4. Tiene una función muy fácil en la aplicación de los retrasos, únicamente es el agente encargado de avisar de la hora de simulación para que los demás sepan si tienen de aplicar algún retraso de tipo 1 o tipo 4.

Recibirá las horas de las que tiene que avisar al leer el documento TriggerCDM.txt, horas que coinciden con alguna Hconocimiento de algún retraso.

El MilestoneTrigger mandara un mensaje a todos los agentes, luego cada agente se preocupara de ver si esa hora le implica tener que realizar algún retraso o no. Al mandar la información de la Hconocimiento, el MilestoneTrigger, agregara un valor en la variable OpcionMensaje siguiendo la tabla 4-14:

Tipo retraso	OpcionMensaje
1	5
4	7

Tabla 4-14: OpcionMensaje en función del tipo de retraso del que informa

Agentes CDM

Los agentes CDM (Airport, GH, ATC, CFMU y AO) son los implicados en la implementación de la propiedad de poder añadir retrasos al programa, funcionan exactamente igual. Serán los agentes encargados de añadir los retrasos a los aviones cuando sea necesario.

La información de los retrasos que aplicará cada agente se recibirá siempre mediante mensajes provenientes del SimulatorInput. Al recibir un mensaje el agente receptor lo escaneara para entender la información recibida.

El escaneado se realizará en la función PonerRetraso, Primero mirará que tipo de retraso tenemos, y en función de esta información escaneara siguiendo el esquema de la figura 4-28:

Tipo 1: Mediante la función *TempsQueRetrasem* buscaremos que tiempo se retrasará en el momento de aplicar el retraso. Acto seguido se guardara la información a la espera de la HoraConocimiento.
Tipo 2: Mediante la función *TempsQueRetrasem* buscaremos que tiempo se retrasará en el momento de aplicar el retraso. Acto seguido se guardara la información a la espera que el futuro avión retrasado llegue al MilestoneConocimiento
Tipo 3: Directamente guarda la información a la espera que el futuro avión retrasado llegue al MilestoneConocimiento

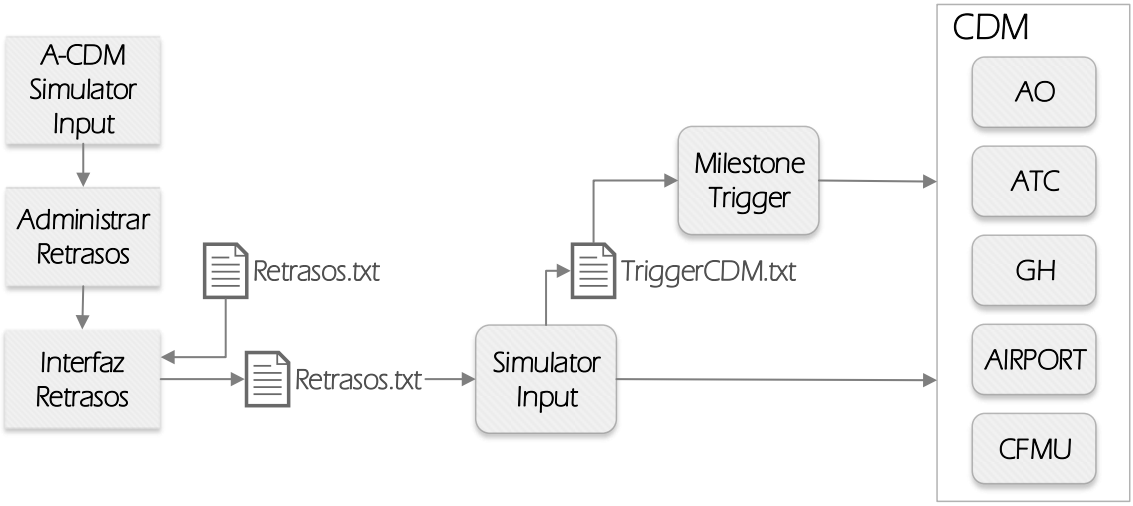


Figura 4-28: Diagrama aplicación retrasos

En la figura 4-28 podemos ver un esquema gráfico de cómo se organiza el programa para definir, crear y aplicar todo lo relacionado con los retraso.

Existen 2 metodos diferentes de aplicar retrasos:

- **Metodo 1**: Multiavion (retrasos tipo 1)
- **Metodo 2**: Un avión (retrasos tipo 2 y topo 3)

Método 1, para retrasos tipo 1
Los retrasos tipo 1, se aplican siempre a una determinada hora, Hconocimiento. Por lo que los agentes que tengan de aplicar estos retrasos necesitaran recibir un mensaje indicando la hora.

Al recibir el mensaje indicando que es la hora, mensaje que vendrá del MilestoneTrigger con OpcionMensaje = 5, primero comprobara si el tiempo que recibe coincide con la Hconocimiento del retraso que tiene que aplicar el agente, podría ser que el aviso de la hora fuera para otro agente, pues el MilestoneTrigger avisa a todos los agentes de que es la hora, pero no dice que agente es el que tiene algún retraso que aplicar.

Si la hora recibida coincide con la Hconocimiento de algún retraso que tiene que aplicar, significa que se tendrá de aplicar el retraso de tipo 1. Entraremos en la función BuscarRetrasosTiempos y miraremos avión por avión si se tiene que retrasar cada uno en función de la Hinicio, Hfinal y MilestoneAfectado. Si el tiempo en el que pasa el MilestoneAfectado está entre la Hinicio y la Hfinal, el modelo coincide con uno de los que se pueden retrasar según ModelosAfectados, y la AO también coincide con alguna de las que se puede

retrasar por AOAfectadas, guardaremos ese avión en la lista de aviones a retrasar.

Acto seguido mediante la función *AplicarRetras* iremos aplicando los retrasos a los aviones seleccionados, y también escribiremos en las incidencias de cada avión la información relativa al retraso aplicado

Método 2, para retrasos tipo 2 y 3
Los retrasos tipo 2 y tipo 3, se aplican de la misma manera, pues conocemos cual será el avión será el retrasado en cada caso, cual es el tiempo que se retrasará y en que milestone se tiene que encontrar el avión para poder aplicarle el retraso.

El agente encargado de aplicar el retraso mirara cada vez que llegue un mensaje con un avión desde el InfoSharing, si el avión del que recibe información está en la lista de aviones de los que tiene que aplicar un retraso, en caso afirmativo mirará el milestone del avión recibido por el InfoSharing, si también coincide aplicara el retraso y lo borrara de la lista de retrasos por aplicar

4.6.2 Reducción Capacidad de Pista

Como se ha explicado anteriormente se quiere habilitar al programa para aplicar cualquier tipo de retraso. Con los definidos anteriormente se considera que un usuario podría aplicar cualquier retraso deseado a los aviones, pero estos retrasos no permitirían simular problemas como una reducción de la capacidad de pista. Por esto, siguiendo más o menos la estructura de los otros retrasos (apartado 4.6.1) se ha definido un cuarto tipo de retraso que representará las reducciones de la capacidad de pista.

De esta manera el programa estará preparado para aplicar reducciones en la capacidad de las pistas por motivos externos. El programa decidirá aplicar un retraso de estas características si recibe la información correspondiente en el documento Retrasos.txt, mirar apartado 4.6.1.1. Las reducciones de pista se consideran a lo largo del programa como retrasos de tipo 4. Estos problemas pueden ser definidos mediante las interfaces para que aparezcan en el documento Retrasos.txt.

Estas reducciones siempre indicarán una nueva capacidad de operaciones por hora en alguna pista. El programa considera como capacidad estándar 30 operaciones/h en cada una de las dos pistas que estamos simulando (una de llegadas y otra de salidas). Al añadir una reducción podemos escoger si se reduce la capacidad en ambas pistas o solo una. No se podrán implementar aumentos de la capacidad de pista, y tampoco se podrá cerrar el funcionamiento del aeropuerto por completo, por lo que se tendrá de escoger un número de operaciones por hora entre 1 y 30 para cada una de las pistas.

A continuación explicaremos el proceso que realiza el programa para aplicar las reducciones de capacidad de pista (retrasos tipo 4).

Simulator Input:
Recordamos que el primer agente involucrado en cualquier retraso es el Simulator Input.

Primero leerá el documento retrasos.txt en la función leerInputretrasos, y se guardara la información de cada línea del documento, acto seguido, mediante la función MandarRetrasos, procederemos a mirar uno por uno los retrasos, y a avisar a los agentes implicados en cada uno de los retrasos. Como siempre que se manda un mensaje entre agentes, se mandara la información en una clase Aircraft, por lo que la información que se mandara para los retrasos contendrá siempre la información del retraso en las incidencias del avión.

Con tal de que los agentes que reciben la información de los retrasos sepan de qué se trata, la OpciónMensaje contendrá un 4, que es el indicador de que se trata de una información relacionada con los retrasos de los aviones.

Cuando encuentre un retraso de tipo 4, el agente a informar será siempre el ATC, pues es el agente encargado de aplicar los retrasos por reducción de la capacidad de pista. El ATC se guardará toda la información relacionada con las reducciones de la capacidad de pista

A continuación añadirá una línea indicando la Hconocimiento de estos retrasos en el documento TriggerCDM.txt. Añadirá una línea que será:

```
Retrasopista   0    Hconocimiento
```

Milestone Trigger:
Como se ha comentado en el apartado 4.6.1.2, el MilestoneTriger solo interviene en los retrasos de tipo 1 y tipo 4, su función es avisar a los agentes de que es la hora de aplicar alguno de estos retrasos. Recibirá las horas de las que tiene que avisar al leer el documento TriggerCDM.txt, horas que coinciden con alguna Hconocimiento de algún retraso.

Recordamos que el MilestoneTrigger avisará de que estamos en una Hconocimiento a todos los agentes de la simulación, aunque en el caso de tratarse de un aviso para una reducción de la capacidad de pista, esta información solo será utilizada por el agente ATC.

El mensaje que manda el MilestoneTrigger variará el valor de OpcionMensaje según el tipo de retraso, mirar tabla 4-15.

Tipo retraso	OpcionMensaje
1	5
4	7

Tabla 4-15: OpcionMensaje en función del tipo de retraso del que informa

Agente ATC:
Es el encargado de aplicar los retrasos a los aviones causados por una reducción de la capacidad de pista.

Cuando reciba un mensaje del SimulatorInput con la información de una reducción de la capacidad de pista solamente se guardará esta información y estará a la espera de que llegue el mensaje del Milestonetrigger que indicará que es la hora de aplicar el retraso.

Los retrasos causados por una reducción de la capacidad de pista se aplicarán siguiendo el siguiente método en el ATC:

Los retrasos tipo 4, se aplican siempre a una determinada hora, Hconocimiento. Por lo que el ATC necesitará recibir un mensaje indicando la hora. Al recibir el mensaje, que vendrá del MilestoneTrigger con OpcionMensaje = 7, comprobara si el tiempo que recibe coincide con la Hconocimiento de la reducción de la capacidad de pista. Para asegurar que no llega una Hconocimiento errónea, y para saber en caso de que tengamos más de una reducción de la capacidad de pista cual es la que se tiene que aplicar en ese instante.

Primero mediante la función *AddSeparationTime* miraremos la nueva distancia mínima en tiempo a la que podrán aterrizar y/o salir dos aviones para que no se supere la máxima capacidad de pista durante las horas en las que tendremos una reducción de la capacidad de pista. En esta función encontraremos las nuevas distancias entre aviones.

A continuación el programa entrara dos veces (una para las llegadas y otra para las salidas) en la función de *PosarSeparacionsReduccioCapacitat.* En esta función miraremos todas las ELDT o ETOT según el caso, e iremos asegurando que el tiempo entre ellos está dentro del mínimo permitido, añadiendo retrasos cuando no se cumplan los requisitos. Una vez estemos fuera de la Hinicio y la Hfinal, la capacidad volverá a ser la nominal.

4.6.3 Pre-departure Sequence

Las políticas de pre-derpartue Sequence explican cómo se organiza el orden de las llegadas y las salidas, quien tiene preferencia y quien tiene que sufrir los retrasos cuando sea necesario. El simulador está preparado para aplicar dos políticas de congestión distintas que explicaremos a continuación:

- **Política 1:** En esta política el primero en entrar en la secuencia tiene preferencia para coger el slot. Por lo que cuando un avión consigue un slot, no se le podrá cambiar su tiempo de salida con motivo de que otro avión quiere salir a la misma hora. El primero que entra en la secuencia será el primero en actuar, por lo que cuando un avión entra en la cola o retrasa su salida/llegada, tendrá de buscar un slot vacío donde posicionarse. En resumen, con esta política un solo avión sufre todo el retraso.
- **Política 2:** Con esta política no importa si eres el primero en entrar en la secuencia, sino que lo que importa es la hora a la que tienes prevista la llegada o la salida. Siempre se da preferencia al primero que quiere actuar, retrasando a los demás en caso de que no se cumplan las distancias mínimas entre slots.

En la tabla 4-16 vemos un ejemplo de cómo quedarían dos secuencias al añadir un avión nuevo “Ejemplo” que quiere actuar a las 11:57. En este ejemplo consideramos que la distancia mínima entre dos aviones tiene que ser de 5 minutos.

Política 1				**Política 2**			
Inicio		**Final**		**Inicio**		**Final**	
Flight 1	11:55	Flight 1	11:55	Flight 1	11:55	Flight 1	11:55
Flight 2	12:00	Flight 2	12:00	Flight 2	12:00	Ejemplo	12:00
Flight 3	12:06	Flight 3	12:06	Flight 3	12:06	Flight 2	12:05
Flight 4	12:17	Ejemplo	12:11	Flight 4	12:17	Flight 3	12:10
Flight 5	12:24	Flight 4	12:17	Flight 5	12:24	Flight 4	12:17
		Flight 5	12:24			Flight 5	12:24

Tabla 4-16: Tabla funcionamientos políticas congestión

4.6.4 Retrasos aleatorios (Distribución estadística)

Con tal de evitar que el programa tenga un comportamiento determinista, y de acercarnos un poco más a la realidad, donde encontramos imprevistos, se ha decidido añadir al programa una opción que permite aplicar una serie de retrasos estadísticos siguiendo una variación normal, con el objetivo de añadir una aleatoriedad a los resultados. Esta variación de los tiempos se lleva a cabo en el momento después de cada uno de los milestones, lo que simula que en la realidad haya un tiempo estimado (con más o menos precisión) y luego el tiempo real difiera levemente.

El usuario podrá decidir activar o desactivar estos retrasos, o seleccionar después de que milestones quiere aplicar esta distribución estadística. En el apéndice de “Manual del Software” encontrará toda la información sobre cómo preparar el programa para que aplique los retrasos deseados en cada simulación.

En caso de que se apliquen retrasos estadísticos, estos solo podrán ser para retrasar tiempos, nunca para adelantar-los. Después de superar un milestone, el agente que este actualizando esta información, será el que llamará a la función *AplicarNormal*, que calculará aleatoriamente si se aplica un retraso en este caso, y en caso afirmativo cuán grande será este.

En la figura 4-29 podemos observar una gráfica de cómo se distribuye una distribución estadística normal. Como no podemos adelantar tiempos, no tendremos en consideración todos los valores negativos

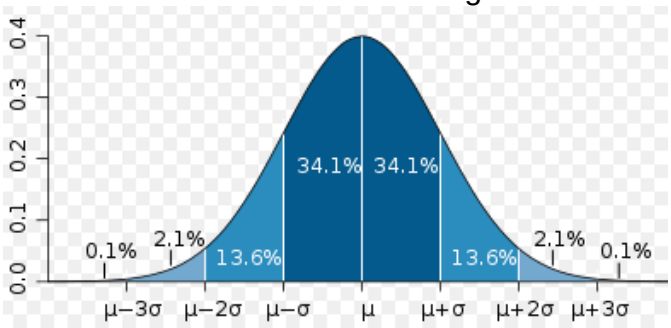


Figura 4-29: Gráfica distribución estadística siguiendo una normal

CAPITULO 5. Desarrollo del software

5.1 *Evolución del programa*

En este apartado hablamos de cómo ha ido evolucionando el simulador, siguiendo las 5 fases especificadas en la fase de definición del programa.

Debido a que el proyecto se ha llevado a cabo por dos alumnos, para poder llevar a cabo una correcta distribución del trabajo y tener el programa ordenado, se optó por asignar tareas a cada uno de los programadores para cada una de las versiones. De esta manera, al acabar ambos alumnos sus tareas, se llevaba a cabo una unión en un solo programa con todo, para cerrar la versión (V1.X) y así no perder nunca el trabajo hecho y tener una base sólida en la que continuar trabajando.

Para explicar con más facilidad la metodología de trabajo seguida para la programación entre dos personas, mostraremos a continuación un diagrama, figura 5-1, donde se puede apreciar con más claridad.

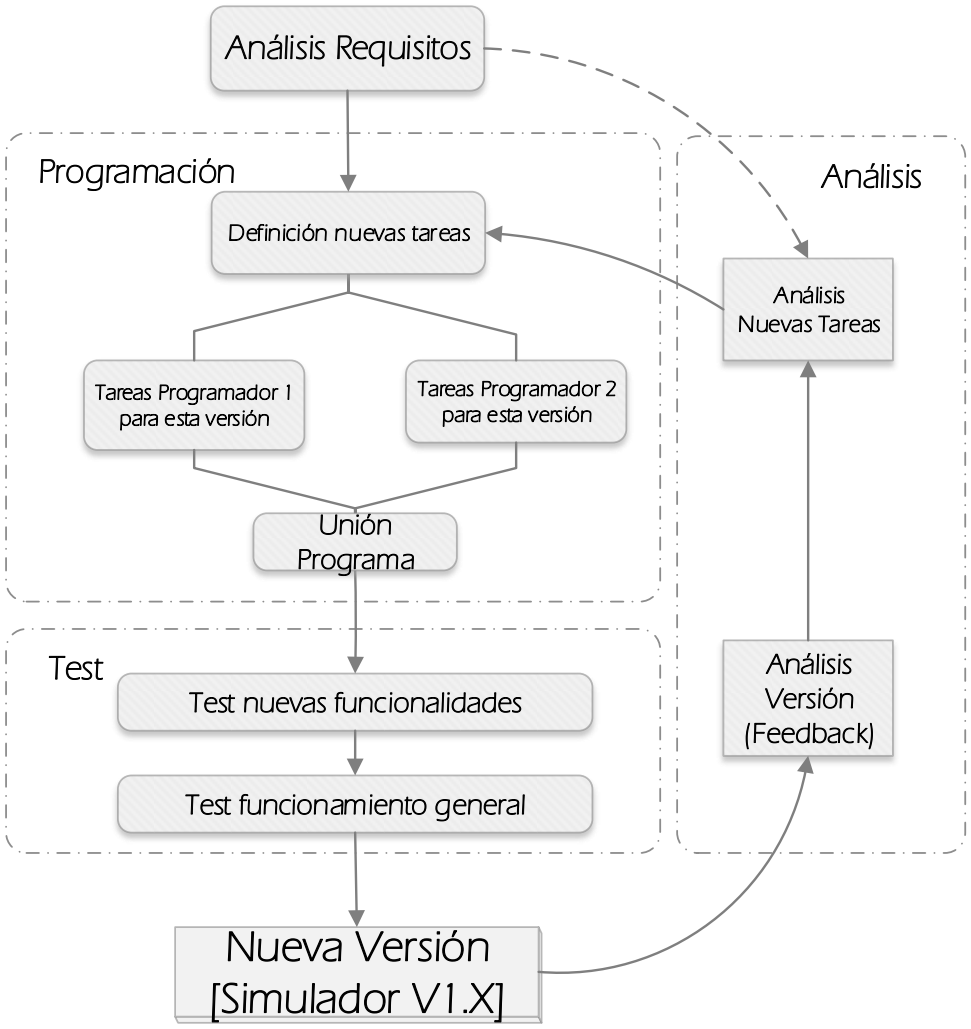


Figura 5-1: Diagrama metodología de trabajo

5.1.1 Versiones

No es necesario entrar mucho en detalles de que tienen las diferentes versiones, ya que lo importante es que la versión final cumple con los requisitos impuestos al inicio del proyecto (actualizados durante el desarrollo). Pero sí que sería interesante ver cómo, aunque se ha intentado seguir las fases lo más posible, al tener que ir actualizando los requisitos y añadiendo funcionalidades, el programa ha ido evolucionando desde una forma más básica y sencilla a una versión final con funcionalidades, gran volumen de tráfico, etc.

A continuación se explican las diferentes versiones, en la tabla 5-1, en las que solo se enumeran los cambios más significativos respecto a la versión anterior; hay que tener en cuenta que una fase importante a la hora de hacer cada una de las versiones ha sido conseguir que las nuevas funcionalidades funcionasen correctamente dentro del conjunto del programa, cosa que no se refleja en las siguientes enumeraciones.

Versiones:

Simulador V0	• Definición Agentes ACDM • Simulación con 1 avión • Simulación solo primer Milestone (Definición de los diferentes tiempos a partir del Input) • Interfaz en Consola Practicas ICARUS, Ivan
Simulador V1	• Agentes SimuladorGrafica • Simulación por Interfaz gráfica
Simulador V1.1	• Simulación MultiAvión (Tráficos de prueba de <10 aviones) • Simulación solo primer Milestone • Agente timming simulación, MilestoneTrigger Inicio TFG (Ivan + Xavi)
Simulador V1.2	• Simulación MultiAvión (Tráficos mayores, <10 aviones, no realistas) • Simulación ciclo ACDM entero • 16 Milestones • Propagación retrasos • Tiempos secundarios • Nueva interfaz gráfica multiavión • Interfaz lista Milestones • Listas agentes Aerolínea y Ground Handling introducidas mediante txt. • Comportamientos en paralelo Agentes MilestoneTrigger y InfoSharing • Tiempos TurnAroundTime a través de txt.
Simulador V1.3	• String “Incidencias” añadido a Aircraft • Funcionalidad Extra: Pre-Departure Sequence

	○ Agente ATC ○ 2 Políticas gestión Pre-Departure Sequence ○ Relación entre retrasos de diferentes aviones, hasta ahora los aviones no se afectaban entre sí. ○ Tanto para despegues como para aterrizajes
Simulador V1.4	• Funcionalidad Extra: Retrasos • Retrasos añadidos por txt • Retrasos simulados por cualquier agente, ejecutados en cualquier momento de la simulación • Funcionalidad Extra: Retrasos estadísticos • Aleatoriedad a la simulación, a cualquier milestone/tiempo de cualquier avión • Posibilidad de ON/OFF la funcionalidad • Configuración parámetros aleatoriedad • Añadidos retrasos a la Pre-departure Sequence • Reducción de capacidad de pista durante una ventana de tiempo • Programa trabaja con aviones que solo aterrizan o solo despegan • Guardar resultados en un txt, para su análisis • Input mejorado • Archivo Airport con los parámetros de la simulación. • Archivo Input con el tráfico aéreo. • Interfaz Entrada (Interfaz Input) • Configuración parámetros sin tocar código • Programa se puede ejecutar en cualquier PC sin tocar código
Simulador V1.5	• Agente DDR2 ○ Lectura datos realesDDR2 • Agente TurnAround (Ej. para proyectos futuros) ○ Nueva conexión mensajes agentes ACDM - Agente externo ○ Nuevo TurnAroundTime mejorado (tiene en cuenta modelo avión y aerolínea • Interfaz Mostrar resultados
Simulador V1.6	• Interfaces para definir y cargar los retrasos • Retrasos ya no vienen obligatoriamente de un txt • Posibilidad guardar retrasos para repetir la simulación en otro momento • Cargar retrasos de otra simulación • Interfaz gráfica durante la simulación mejorada • Posibilidad parar la simulación en cualquier momento y guardar los resultados

	• Actualización Interfaz Input (Abrir interfaces para administrar los retrasos) • Interfaz Resultados mejorada • Posibilidad cargar otro set de resultados, a la vez que se muestran los de la simulación • Comparación resultados ambas simulaciones
Simulador V1.7	• Interfaz Input mejorada • Posibilidad pasar a analizar resultados sin necesidad de simular (carga de resultados guardados previamente) • Interfaz Análisis resultados • PLOT: Análisis de los aviones retrasados durante la simulación • Análisis de resultados en cualquiera de los tiempos del Aircraft • Posibilidad filtrar resultados a analizar (por aerolíneas/Modelo) • Plot: aviones retrasados vs tiempo (área = minutos retrasados totales) • Visión localizada de los retrasos. Elegir ventana de tiempo del análisis
Simulador V2	• Varios errores corregidos • Robustez en las interfaces Versión entregada. Fin TFG

Tabla 5-1: Evolución versiones del Simulador

Es necesario comentar que, como se puede ver en la tabla, las primeras versiones provienen de un trabajo de prácticas del alumno Ivan Garcia, y no son propiamente del TFG. No son relevantes en el proyecto pero importante explicar desde donde se había cogido el programa y que se viera que esas versiones estaban muy atrasadas respecto a las siguientes, programadas por los dos alumnos que realizan este trabajo.

5.2 *Problemas durante el desarrollo*

Durante el desarrollo del simulador nos hemos enfrentado a una gran cantidad de problemas y adversidades que se han tenido de ir superando con tal de poder ir avanzando y mejorando el programa. Algunos de estos problemas se han podido solucionar en cuestión de minutos, pero otros han sido los causantes de verdaderos quebraderos de cabeza. A continuación procederemos a explicar algunos de estos problemas que hemos encontrado. Para cada problema explicaremos cual era realmente el problema, y como se ha solucionado.

- **Comportamientos del MilestoneTrigger:** Desde el primer momento teníamos clara la idea de que el agente MilestoneTrigger sería el encargado de avisar de los milestones realizados por los aviones, i también de ordenar a la vez la lista de milestones tras cada actualización de algún tiempo de algún avión. El problema que

teníamos, es que necesitábamos que el MilestoneTrigger realizara dos acciones relacionadas a la vez. Nos dimos cuenta que los comportamientos (behaviours) simples que usamos en la mayoría de agentes no nos servían para este problema. Tras investigar más a fondo la información relacionada con los comportamientos se decidió implementar 2 comportamientos paralelos, que permiten al agente realizar dos tareas a la vez sin problemas. Se implementaron 2 comportamientos, uno se encarga de actualizar la lista de milestones, y el otro de mandar los mensajes de un nuevo milestone cada vez que pasa el tiempo de timer.

- **Retrasos:** El poder aplicar retrasos seleccionados por el usuario no ha sido una tarea fácil. El problema vino en que esta opción no entraba dentro de los planes iniciales. Decidimos implementar esta opción cuando el programa ya estaba en una fase de bastante avanzada, y al ser una tarea que afecta a muchos agentes, puede afectar a todos los tiempos y no siempre se aplica de la misma manera (tipo 1, 2, 3 y 4), causo que se necesitaran realizar muchas modificaciones en todo el código con tal de dejar el programa preparado para estas situaciones
- **Agente Externo Turn Around:** Como ya se ha explicado, el agente Turn Around tiene únicamente una función, que es la de devolver el tiempo de turn around de un avión al agente GH. Esto causo problemas en el agente GH, pues teníamos de crear 2 blockingReceivers, uno para recibir mensajes de actualizaciones de tiempos y otro para recibir las respuestas del Turn Around. Al principio se intentó solucionar este problema creando dos comportamientos en paralelo igual que el agente MilestoneTrigger, pero aunque cada agente tenía un blockingReceive todos los mensajes entraban en el mismo comportamiento. En ese momento vimos que la solución no eran 2 comportamientos en paralelo, sino crear un solo comportamiento con 2 blockingReceive, i una lista de mensajes recibidos en la memoria del agente, donde guardamos los mensajes cuando estos no son los que esperamos en ese instante para poder continuar procesando información.

5.3 *Validación y Verificación*

El proceso de validación y verificación (V&V) está compuesto de un conjunto de procesos de análisis y comprobación que aseguran que el software que se desarrolla está acorde a su especificación y cumple con las necesidades del cliente.

Aunque son términos que en ocasiones se utilizan indistintamente, verificación y validación, representan conceptos distintos.

La verificación busca comprobar que el software está de acuerdo con su especificación. Es decir que el sistema cumple con los requisitos, funcionales y no funcionales, que se le han especificado. En otras palabras busca determinar si estamos construyendo el simulador correctamente [12].

Por otra parte la validación es un proceso más general que busca comprobar si el software cumple con las expectativas del cliente. En otras palabras busca determinar si estamos construyendo el simulador correcto, o por el contrario se está construyendo un software que nada tiene que ver con las expectativas del cliente.

Al tratarse de un software informático es importante comprobar después de cada modificación por pequeña que sea en el código, que este sigue funcionando correctamente aplicando las nuevas variaciones. Por este motivo a continuación expondremos una serie de comprobaciones que creemos que se deben realizar después de cualquier modificación. Evidentemente dependiendo de las modificaciones realizadas, el software necesitará de nuevas validaciones y verificaciones para verificar su funcionamiento, incluso llegando a darse el caso que las verificaciones que se expondrán a continuación dejarán de ser válidas.

Es importante tener en consideración que a la hora de realizar una validación del simulador, se tiene que simular un caso muy simple del cual predecir los resultados antes de simular, de manera que podremos saber si todo sigue funcionando correctamente. Con tal de simplificar los casos que simularemos para las validaciones diremos a los agentes AO, GH y ATC que no apliquen variaciones aleatorias, por lo que pondremos el parámetro *variacionesnormales*=false en cada uno de estos agentes.

- **Lectura documento Eurocontrol**: La primera comprobación será siempre la de asegurar que el agente sigue siendo capaz de leer el documento DDR2 de Eurocontrol y a la vez escribir el documento Input.txt. Para realizar esta comprobación le entraremos un documento DDR2 que contenga la información del tráfico aéreo europeo, pero solo durante un periodo de una o dos horas, con tal de disminuir la cantidad de vuelos del documento. A continuación comprobaremos que el documento que ha escrito Input.txt contiene la información estructurada correctamente (Mirar apartado 4.5.1, donde se explica la información contenida en Input.txt).

Una vez realizada la primera comprobación procederemos a escoger los valores del Input.txt que nos interese en cada momento. Para poder seguir con estas comprobaciones deberemos entrar dentro del código del agente *SimulatorInput* y en la función d*efinirRutas* modificaremos la ruta del Input.txt a Input2.txt, tal y como vemos en la figura 5-2:

```
public void definirRutas(Aircraft avion){

    String ruta = rutaArchivos(avion);

    rutaInput = ruta + "Input2.txt"; //Modificado de Input.txt a Input2.txt
    rutaFP = ruta + "FlightPlans.txt";
    rutaMilestones = ruta + "TriggerCDM.txt";
    rutaAO1 = ruta + "AO1.txt";
    rutaAO2 = ruta + "AO2.txt";
    rutaAO = ruta + "VuelosAerolinea.txt";
    rutaFPS = ruta + "FlightPlansSalidas.txt";
    rutaRetrasos = ruta + "Retrasos.txt";

}
```

Figura 5-2: Código modificado para realizar las validaciones

Desde este momento el simulador siempre simulará los aviones que se encuentren en el documento Input2.txt sin tener en cuenta el documento de entrada SO6 que se entre en cada caso.

A continuación explicaremos como modificaremos cada vez el documento input2.txt y que miraremos para verificar todas y cada una de las comprobaciones que decidamos realizar:

- **Caso General:** Información en el documento Input2.txt:

```
2    EZY87ER   A319 LEBL LFPG 1500 -
1    EZY87ER   A319 LIMC LEBL 1030 1134 1139 1144
```

Tenemos un solo avión escrito de manera que encontramos primero la salida y luego la llegada, para comprobar que el orden de entrada de los dato en el Input.txt no afecta, y de paso aseguramos que el simulador funciona en el caso general.
Si miramos la figura 5-3 del *A-CDM Resultados* encontraremos:

N	CALLSIGN	STATUS	MODEL	ETOT	ELDT	EIBT	EOBT	ETOT	Milestone
1	EZY87ER	DEPARTED	A319	1030	1144	1159	1440	1500	16

Figura 5-3: Resultado validación Caso General

- **Solo llegadas y solo salidas**: Información en el documento Input2.txt:

```
2    EZY87ER   A319 LEBL LFPG 1500 -
1    EZY87ER   A319 LIMC LEBL 1030 1134 1139 1144
2    RYR300    A319 LEBL LFPG 1600 -
1    IBE700    A319 LIMC LEBL 0930 1034 1039 1044
```

Tenemos el avión utilizado anteriormente más dos aviones, uno que solo llega y otro que solo sale. Miraremos que funcione para el caso general, pues aún no tenemos ningún tipo de problema por congestión de pista, pues los dos aviones llegan con diferencias de tiempo muy considerables.
Si miramos la figura 5-4 del *A-CDM Resultados* encontraremos:

N	CALLSIGN	STATUS	MODEL	ETOT'	ELDT	EIBT	EOBT	ETOT	Milesto...
1	IBE700	INBLOCK	A319	0930	1044	1059	-	-	8
2	EZY87ER	DEPARTED	A319	1030	1144	1159	1440	1500	16
3	RYR300	DEPARTED	A319	-	-	-	1540	1600	16

Figura 5-4: Resultado validación solo llegadas y salidas

- **Pista Congestionada**: Información en el documento Input2.txt:

```
2    EZY87ER   A319 LEBL LFPG 1559 -
2    RYR300    A319 LEBL LFPG 1600 -
1    VLG870    A319 LIMC LEBL 1030 1134 1139 1144
1    IBE700    A319 LIMC LEBL 0930 1134 1139 1145
```

Tenemos 4 aviones, dos de solo llegada y dos de solo salida, estamos interesados en ver como procesa el simulador este caso, pues los aviones tienen previstas las llegadas y salidas con muy poca diferencia de tiempo entre ellos.

Como aquí entrará en acción el predeparture sequence, es importante tener en cuenta la *politicaCongestio* seleccionada en el ATC a la hora de simular.

Cuando ***politicaCongestio* = 1:** Damos preferencia al avión que se define primero, por lo que el orden de las llegadas tendrá de ser: IBE700 seguido de VLG870, mientras que el orden de salidas será EZY870ER seguido de RYR300.

Si miramos la figura 6-5 del *A-CDM Resultados* encontraremos:

N	CALLSIGN	STATUS	MODEL	ETOT'	ELDT	EIBT	EOBT	ETOT	Milesto...
1	IBE700	INBLOCK	A319	0930	1145	1200	-	-	8
2	VLG870	INBLOCK	A319	1030	1147	1202	-	-	8
3	EZY87ER	DEPARTED	A319	-	-	-	1539	1559	16
4	RYR300	DEPARTED	A319	-	-	-	1541	1601	16

Figura 5-5: Resultado validación politicaCongestio = 1

Cuando ***politicaCongestio* = 2:** Damos preferencia al avión que tiene intención de llegar/salir primer, sin importar si es el primero en ser definido, por lo que el orden de las llegadas tendrá de ser: VLG870 seguido de IBE700, mientras que el orden de salidas será EZY870ER seguido de RYR300.

Si miramos la figura 5-6 del *A-CDM Resultados* encontraremos:

N	CALLSIGN	STATUS	MODEL	ETOT'	ELDT	EIBT	EOBT	ETOT	Milesto...
1	IBE700	INBLOCK	A319	0930	1146	1201	-	-	8
2	VLG870	INBLOCK	A319	1030	1144	1159	-	-	8
3	EZY87ER	DEPARTED	A319	-	-	-	1539	1559	16
4	RYR300	DEPARTED	A319	-	-	-	1541	1601	16

Figura 5-6: Resultado validación politicaCongestio = 2

En caso de que en un futuro se programen más políticas de congestión tendríamos de demostrar su funcionamiento haciendo alguna prueba para verificarlo.

- **Comprobación Retrasos (tipo 1, 2 y 3):** Información en el documento Input2.txt:

```
2    EZY87ER   A319 LEBL LFPG 1500 -
1    EZY87ER   A319 LIMC LEBL 1030 1134 1139 1144
```

Tenemos de verificar que los retrasos se aplican correctamente, por lo que mediante la interfaz retraso y usando un solo avión lo expondremos mediante 3 simulaciones a un retraso en cada caso. En la primera simulación de tipo 1, en la segunda de tipo 2 y en la tercera de tipo 3. Comprobaremos en la tabla de *A-CDM Resultados* que el retraso seleccionado se aplica correctamente.

- **Comprobación Retrasos (tipo 4). Reducción Capacidad de pista:** Información en el documento Input2.txt:

```
2    EZY87ER   A319 LEBL LFPG 1600 -
2    RYR300    A319 LEBL LFPG 1605 -
1    VLG870    A319 LIMC LEBL 1030 1134 1139 1150
1    IBE700    A319 LIMC LEBL 0930 1134 1139 1145
```

En este caso tenemos 2 aviones que llegan y que salen con una diferencia de tiempo de 5 minutos entre ellos, por lo que en situaciones normales esto no será un problema, pues la capacidad por hora de la pista implica que puedan llegar y salir aviones con intervalos de tiempo de 2 minutos entre ellos. Pero aplicaremos una reducción de la capacidad de pista para comprobar el buen funcionamiento de los retrasos de tipo 4 en el programa.

Para aplicar esta reducción escribiremos en el documento Retrasos.txt:

```
4 1100 2000 1000 5 5 Ejemplo
```

I a continuación le diremos al programa que tenga en cuenta estos retrasos para la simulación entrando en el botón de *Administrar Retrasos* i a continuación indicándole que lea el documento de retrasos.txt.

Eso implica que el simulador aceptara una llegada i una salida cada 12 minutos (5 aviones por hora). Con una *politicaCongestio=1* obtendremos la figura 5-7 en el *A-CDM Resultados*:

N	CALLSIGN	STATUS	MODEL	ETOT'	ELDT	EIBT	EOBT	ETOT	Milesto...
1	IBE700	INBLOCK	A319	0930	1145	1200	-	-	8
2	VLG870	INBLOCK	A319	1030	1157	1212	-	-	8
3	EZY87ER	DEPARTED	A319	-	-	-	1540	1600	16
4	RYR300	DEPARTED	A319	-	-	-	1552	1612	16

Figura 5-7: Resultado reducción capacidad de pista, politicaCongestio = 1

Una vez realizadas estas comprobaciones con éxito podemos suponer que el software funciona en términos generales. Pero aún podríamos tener algún error en situaciones muy concretas, de manera que podría ser que necesitáramos realizar nuevas comprobaciones con tal de verificar nuevos procedimientos que se pudieran llegar a programar, o podría ser que alguna de las validaciones explicadas anteriormente dejara de ser útil en una futura versión del programa.

Al terminar el proceso de Validación y Verificación tenemos de volver a modificar el agente *SimulatorInput* de manera que la función *definirRutas* vuelva a leer el documento de salida del agente DDR2 (**Input.txt**) en lugar del documento Input2.txt con tal de volver a realizar simulaciones completas

5.4 *Explicación ejecución del programa*

Debido a que se ha realizado un entorno de simulación basado en agentes, se ha llegado a un programa descentralizado y con comunicaciones asíncronas constantes.

Por esta razón es necesario este apartado donde se explica la manera en la cual se ejecuta el programa. Una explicación del orden en el que ejecutan los agentes, que mensajes envían (y cuando) y cómo reaccionan.

Para esta explicación es importante dividir la ejecución del programa en 3 partes totalmente diferenciadas, independientes y que no se ejecutan a la vez:

Inicio Programa (Input Simulación): El programa define los parámetros iniciales, carga y genera los archivos necesarios para simular, e inicia la simulación
Simulación A-CDM: El programa simula el tráfico aéreo.
Fin Programa (Output Simulación): El programa muestra los resultados de la simulación, guarda los archivos de resultados y abre las interfaces de análisis.
Las fronteras de esta división del programa son un poco difusas, ya que en la parte de Fin de Simulación (output) podemos encontrar dos excepciones concretas:

- Generación del documento MilestoneResultado.txt. Se trata de un documento Output que nos informa de los milestones que se han ejecutado durante la simulación. Debido a que el usuario puede parar la simulación en cualquier momento, este documento no es generado en la parte de Output, sino que durante la simulación el agente MilestoneTrigger lo va escribiendo.
- División de Fin Simulación (Output). Debido a que el programa ofrece la posibilidad de analizar resultados desde archivo, sin necesidad de simular, encontramos que es posible pasar de Input a Output sin pasar por la Simulación ACDM. Cuando esto pasa, la ejecución de Output cambia, solo ejecuta la parte de mostrar y analizar resultados, ya que no necesita generar ningún archivo.

Una vez tenemos claras las diferentes partes de la ejecución y las excepciones, podemos ver cómo queda el esquema general de ejecución del programa, figura 5-8:

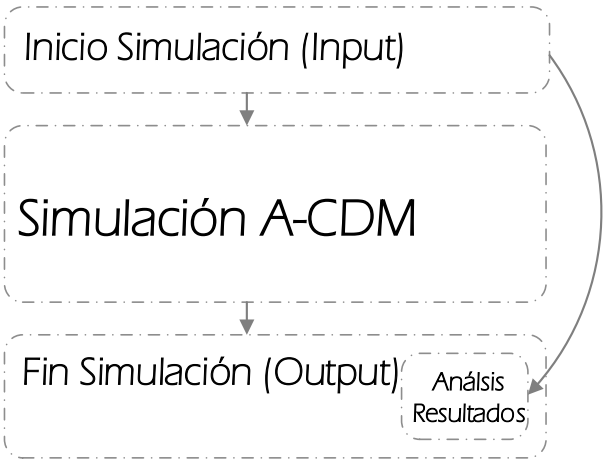


Figura 5-8: Esquema Ejecución Programa

Ahora que tenemos esquematizada la ejecución es necesario explicar cada una de las partes con más detalle.

Para ello nos serviremos de esquemas temporales. Para ver como se realiza la ejecución de cada parte en estos esquemas hay que entender que esta esquematizado de manera temporal descendente. El esquema nos mostrará lo que pasa durante la ejecución, ordenado de manera que lo que está arriba pasa antes de lo que tiene justo debajo.

Esta manera de esquematizar nos ayudará a entender la ejecución, puesto que veremos que aparecen comportamientos de agentes, conexiones con interfaces y recepción y envió de mensajes.

5.4.1 Ejecución Inicio Programa

En esta parte de inicio del programa se realizarán las siguientes tareas:

- Definición de los parámetros iniciales
- Carga y lectura de los archivos input
- Generación de los archivos input de la simulación
- Broadcast de Parámetros Iniciales y Archivos input.
- Administración de retrasos de la simulación
- Inicio de la simulación

Para realizar estas tareas, en esta parte de la simulación actúan dos agentes. Simulator Input se encarga de administrar los parámetros iniciales, cargar los archivos input, administrar los retrasos de la simulación y enviar los mensajes para iniciar la simulación. Por otra parte, el agente DDR2 carga y genera el archivo input del tráfico aéreo a partir del archivo del DDR2.

Para ver lo que sucede de una manera más esquematizada podemos observar la figura 5-9:

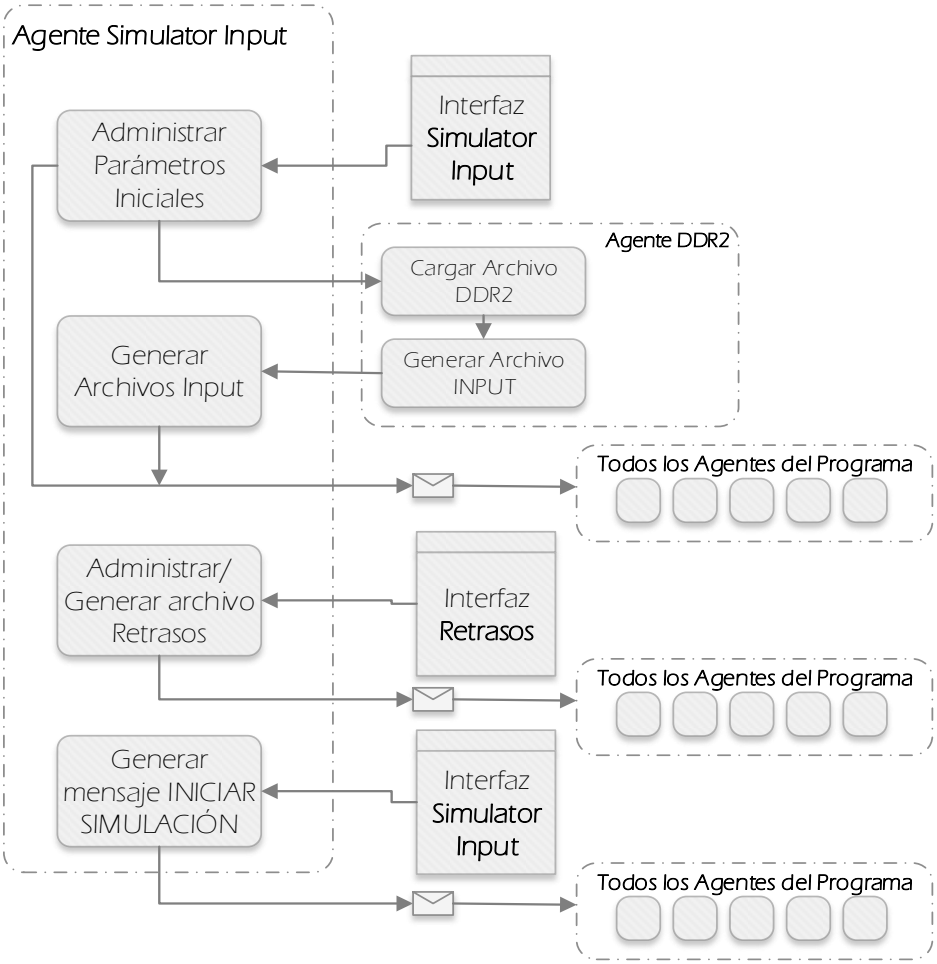


Figura 5-9: Esquema Temporal Inicio Programa

En este diagrama podemos ver que lo primero que hace el programa es administrar los parámetros iniciales de la simulación a través de la interfaz Simulator Input. Una vez se han cargado estos parámetros y DDR2 los recibe, este procede a cargar y generar el archivo input de tráfico aéreo; con el que Simulator Input genera los archivos input de la simulación, que después envía a todos los agentes.

Una vez se han generado todos estos archivos input y se han compartido los parámetros iniciales, Simulator Input administra los retrasos de la simulación a través de su interfaz y una vez definidos los comparte con todos los agentes del programa.

Finalmente, el usuario decide iniciar la simulación a través de la interfaz.

Todo este proceso no sería necesario en caso de que el usuario decidiera pasar directamente a analizar resultados. En tal caso el agente Simulator Input cargaría la ruta donde se encuentran dichos resultados y enviaría un mensaje para que se iniciara la ejecución de análisis de resultados.

5.4.2 Ejecución Simulación A-CDM

Esta es la ejecución más grande e importante de las que se han enumerado. Se lleva a cabo la simulación del tráfico aéreo hasta el final del tiempo de simulación.

La peculiaridad de esta simulación es que se ejecuta como un bucle, donde un timmer decide el momento de iniciar cada una de las simulaciones.

También encontramos un bucle en el hecho de que hay mensajes entre los agentes CDM y el agente InfoSharing hasta que todos los aviones han actualizado todos los tiempos en ese milestone.

En el diagrama de la figura 5-10 podemos ver claramente el bucle interno de la simulación (Agente CDM – Agente InfoSharing) y donde actúa el timmer.

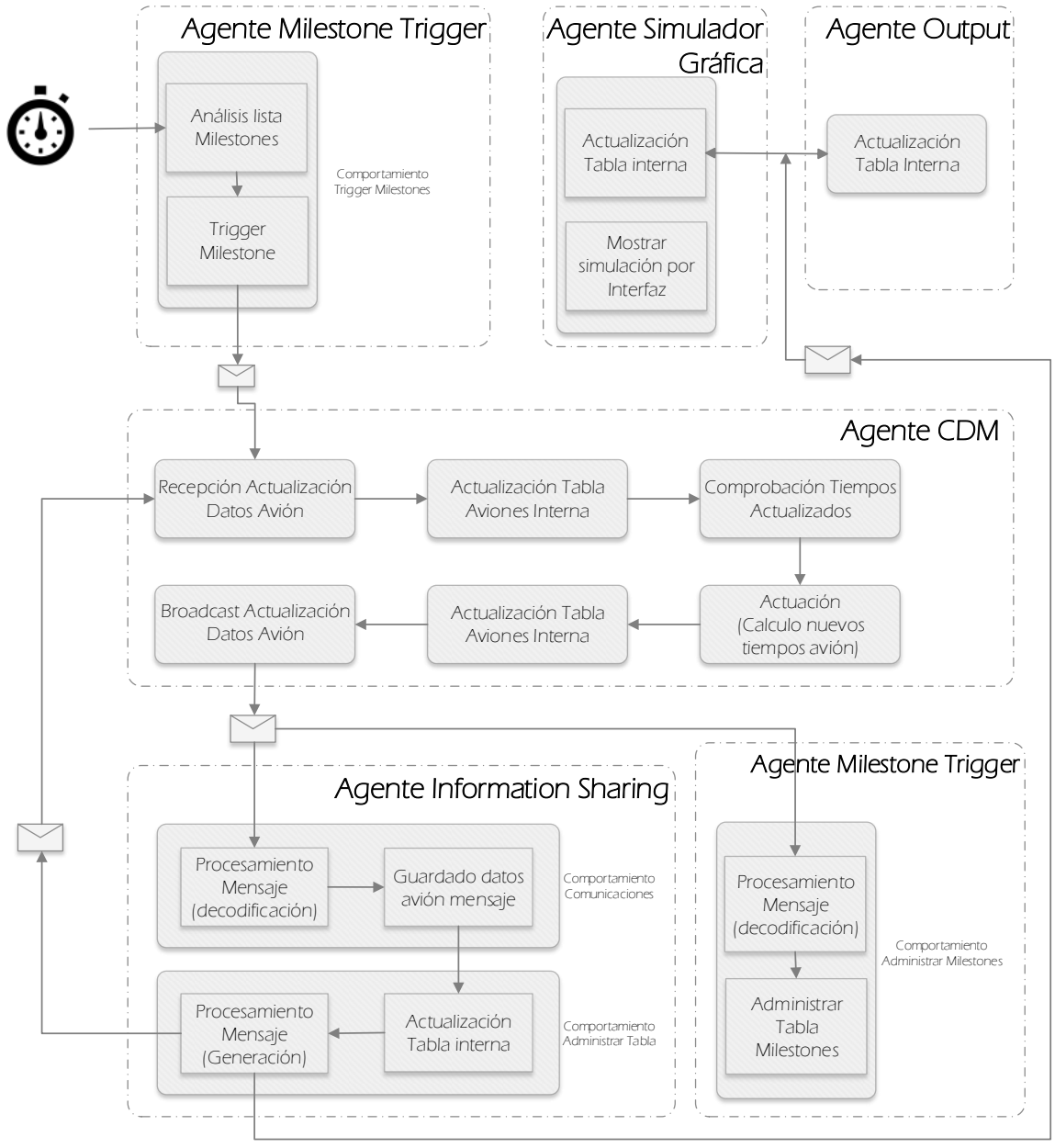


Figura 5-10: Esquema temporal simulación A-CDM

Al tratarse de un diagrama temporal y sabiendo en que momento de la simulación se produce la comunicación entre el agente InfoSharing y los agentes Simulador Gráfica y Output, estos deberían estar en la parte inferior. Pero para obtener un diagrama más compacto y al ver que este cambio no afectaba a la comprensión del mismo, hemos optado por ponerlos arriba.

En el diagrama podemos apreciar claramente como los agentes InfoSharing y MilestoneTrigger cuentan con dos comportamientos ejecutándose en paralelo. Esto es debido al gran número de tráfico de mensajes que deben administrar.

Para simplificar el diagrama no se ha dibujado la relación entre el comportamiento del agente Milestone Trigger que hay al principio con el del final. Estos dos comportamientos están directamente relacionados porque son comportamientos de un mismo agente, por esa misma razón la comunicación entre ellos es directa. Concretamente, la tabla que administra y ordena el comportamiento *Administrar Milestones* es la misma que el comportamiento *Trigger Milestones* lee y analiza.

El Agente CDM del diagrama representa los 5 *agentes ACDM* que encontramos en el programa. Es posible juntarlos en este diagrama debido a que la elección de quien recibe cada uno de los mensajes de Milestone Trigger depende únicamente del milestone que se esté ejecutando. Además, el mensaje que recibe del agente InfoSharing es un Broadcast que reciben los 5 agentes en el programa.

Se puede apreciar claramente que la simulación de cada uno de los milestones empezará en cada tic del timmer, cuya velocidad la decide el usuario.

La razón de que encontramos un bucle dentro de cada simulación de un milestone es que en la ejecución de cada uno de estos milestones encontramos que se actualizan diferentes tiempos de diferentes aviones. Siguiendo un poco la simulación podemos ver como un agente CDM recibe una actualización de tiempos, actualiza su tabla, analiza los tiempos y calcula y actualiza otros tiempos del avión; esto hace que al enviar esta actualización al InfoSharing, este tenga que enviar un broadcast a todos los agentes CDM informando de la actualización, que a su vez puede provocar que haya una nueva actualización de tiempos por parte de otro agente CDM, cosa que hace que el bucle vuelva a empezar.

La velocidad de procesamiento del programa hace que este bucle InfoSharing-CDM se termine antes de iniciarse la simulación del siguiente milestone.

5.4.3 Ejecución Fin Programa

En el programa hay 3 caminos diferentes para llegar a esta parte de la ejecución:

1. Simulación completa del tráfico aéreo.
2. Simulación parcial del tráfico aéreo.
3. Cargar resultados de una simulación previa.

En el siguiente diagrama, figura 5-11 vemos como se ejecuta el fin del programa al llegar por cualquiera de los tres caminos. La ejecución que se inicia con el Milestone Trigger corresponde a una simulación completa, la ejecución iniciada por Simulador Gráfica corresponde a una simulación parcial y la ejecución empezada por Information Sharing corresponde al tercer camino en el que cargamos los resultados de una simulación ya realizada.

Como podemos ver en la figura 5-11, la causa de que se inicie la primera ejecución es que Milestone Trigger ha llegado al final de su lista de milestones, por tanto ha llegado al final de todo lo que se tenía que simular. Por otra parte, la simulación parcial se debe a que, a través de la interfaz que muestra la simulación, el usuario ha decidido cortar la simulación en ese punto y observar los resultados.

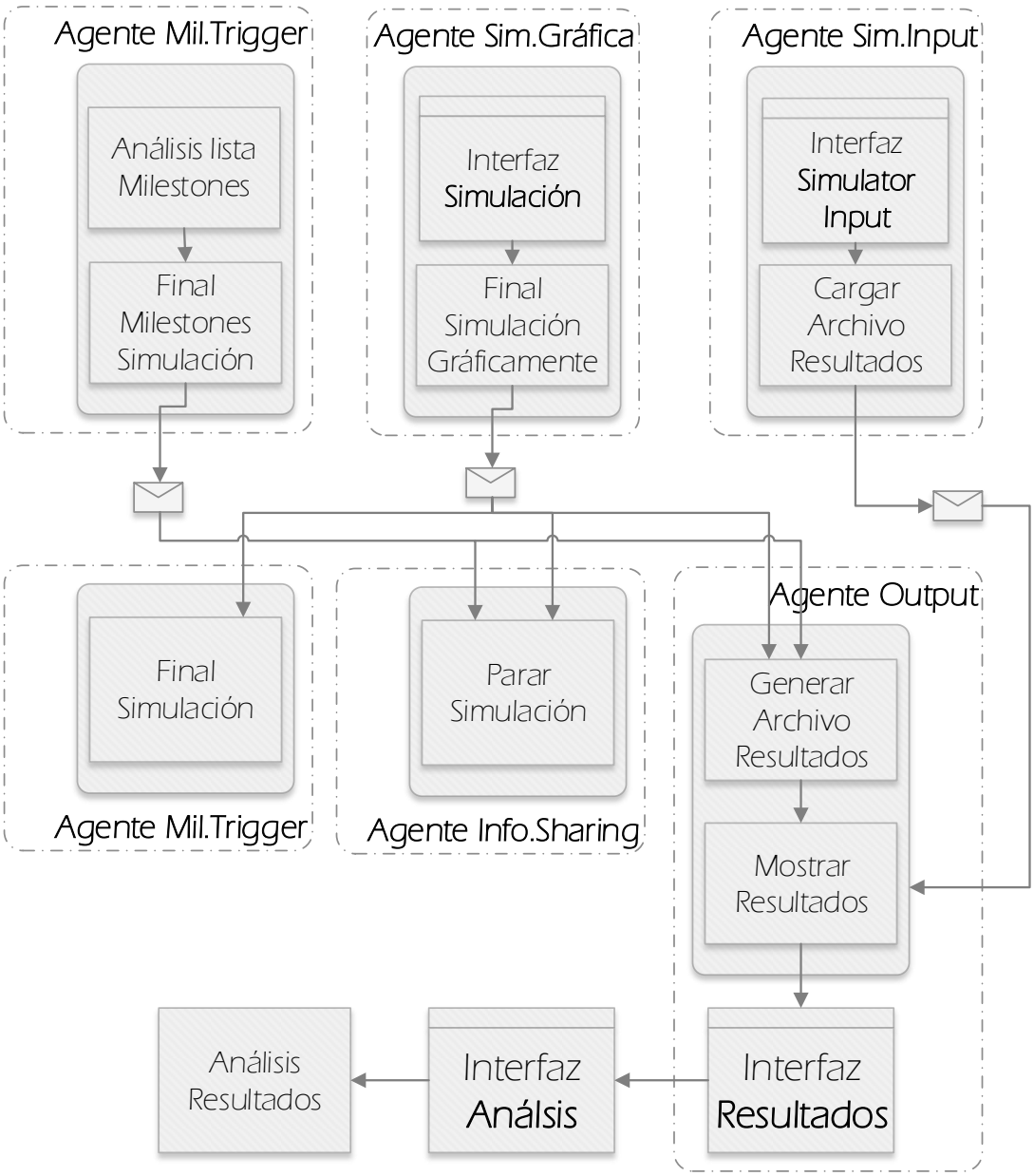


Figura 5-11: Esquema ejecución Fin Programa

Al final del diagrama también se muestra la ejecución de un análisis de datos, este sería el punto en común de los 3 caminos. Se puede analizar resultados después de una simulación, completa o parcial, y a través de cargar resultados desde la interfaz del Simulator Input; en cualquiera de los 3 análisis el programa da la posibilidad de analizar y comparar con otro set de resultados.

5.5 *Cálculo de tiempos de Milestones*

Una de las bases del programa es el cálculo de todos los tiempos en los que un avión iniciará cada uno de los milestones definidos en todo momento. A partir de estos tiempos se podrá mantener una visión global de lo que está sucediendo en el aeropuerto en cada momento y realizar las acciones necesarias ante situaciones futuras.

El programa es capaz de, a partir de los datos que nos brinda el input de tráfico aéreo (tiempos de vuelo, despegues y aterrizajes), calcular todos los tiempos que son necesarios para el ciclo A-CDM. En el capítulo 2 se han explicado estos tiempos y a que milestone pertenecen. Por esta razón a continuación solo hay una pequeña tabla enumerándolos, tabla 5-2, para recordarlos y poder entender mejor los apartados siguientes:

MILESTONE	TIEMPO
1	ETOT'-3horas
2	ETOT'-2horas
3	ETOT'
4	TFIR
5	TAPP
6	ELDT
7	EIBT
8	ACGT
9	Confirmación EOBT
10	Definición TSAT
11	ASBT
12	ARDT
13	ASRT
14	TSAT
15	EOBT
16	ETOT

Tabla 5-2: Acción producida al realizar un milestones

El programa tendrá dos maneras de (o situaciones donde) cambiar estos tiempos:

Definición del avión en el sistema: en la situación de definir por primera vez el avión en el sistema, todos los agentes utilizarán su inteligencia interna para definir y dar valor a los diferentes tiempos a partir de los datos del Input del tráfico aéreo y de los demás tiempos ya definidos.

Debido a que este proceso depende inicialmente de los datos del tráfico aéreo (y de los parámetros que afectan a la inteligencia de los agentes), hemos definido esta situación como "INPUT → Todos los tiempos".

Propagación de un retraso: durante la simulación hay continuas actualizaciones de los tiempos de los aviones, producidas por todo tipo de retrasos y situaciones que afectan a uno o más aeronaves. Estas situaciones hacen que podamos tener actualizaciones de cualquiera de los tiempos en cualquier momento y que estos retrasos, aparte de propagarse en los tiempos de un mismo avión, afecten a los vuelos que están a su alrededor.

Para un comportamiento realista de la propagación de estos retrasos se han definido una serie de relaciones entre los tiempos de un avión. Por ejemplo al retrasar el ELDT (Tiempo estimado de aterrizaje), el tiempo EIBT (Tiempo estimado de llegada a Gate) se verá afectado y también se retrasará.

Este proceso se ha definido con el nombre "Retraso→Propagación Retraso" en la memoria, y se explicará más a fondo a continuación.

El agente InfoSharing realizará un broadcast de la información de un avión cada vez que reciba una actualización en alguno de los parámetros de un avión. Ya sean actualizaciones porque se està definiendo el avión, porque este està sufriendo algún retraso en alguno de sus tiempos o porque acaba de realizar algún milestone.

A continuación se explica el proceso y las relaciones establecidas para definir y actualizar todos los tiempos de un avión utilizados en el programa.

5.5.1 Definición Tiempos Avión

Durante una simulación completa, los agentes se avisan constantemente de cualquier cambio realizado en los tiempos de un avión con el objetivo de que a partir de la nueva información los otros agentes puedan calcular cuanto antes los efectos que causará cada variación en los otros tiempos.

La primera vez que se calcularán estos parámetros es cuando se define el avión. En el momento en que un avión realiza el primer milestone, este tiene que definir muchos tiempos, ya que solo se conocen los datos que venían sobre dicho avión en el documento input.txt. En ese momento los agentes empezaran a calcular nuevos tiempos y a compartir la nueva información recopilada después de los cálculos realizados en cada período.

A continuación mostraremos un esquema, figura 5-12, en donde vemos el proceso que sigue un avión para definir todos los tiempos una vez este realiza el primer milestone y solo se conoce de él la información contenida en Input.txt.

En cada período podemos observar que se calculan nuevos parámetros, nuevos tiempos, estos parámetros son calculados por los agentes CDM utilizando la información disponible del avión en el período anterior. En cada período tiene más información y puede calcular nuevos parámetros. Después de 4 períodos el simulador dispone de la información necesaria para definir todos los tiempos en

los que el avión realiza alguna acción, y por lo tanto ya dispone de la información necesaria para definir todos los tiempos de milestone.

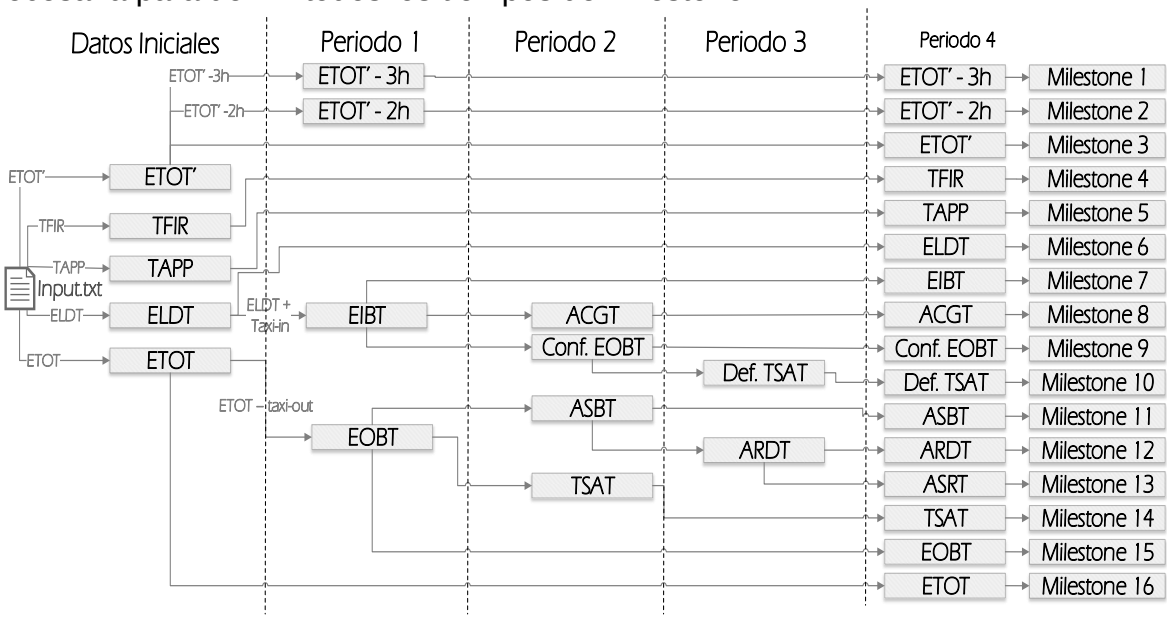


Figura 5-12: Esquema de definición de los tiempos de un avión

Si considerásemos la premisa de que no se conoce ningún retraso para el avión en el momento de definir sus parámetros, al terminar estos 4 periodos, ya sabríamos todos los movimientos que realizará el avión desde que sale del aeropuerto de origen, llega al nuestro y después de realizar el turn around vuelve a salir. Todos estos cálculos se realizan 3 horas antes de que el avión salga del aeropuerto de origen, en el momento de realizar el milestone 1, que es el momento en que el avión apareze por primera vez en el simulador.

5.5.2 Propagación Retrasos

Como ya se ha explicado, el programa acepta retrasos, por lo que en cualquier momento un agente puede recibir la orden de retrasar un tiempo en concreto, si esto pasara, los demás agentes tendrán de ser informados y deberán calcular si ese retraso afecta a alguno de los otros tiempos del avión afectado. Cualquier tiempo puede ser retrasado en cualquier momento, pero un tiempo nunca resultará adelantado.

El programa debe estar preparado para aceptar cualquier retraso en cualquier momento, por eso es importante que se cree una malla de comprobaciones entre los diferentes milestones como la de la figura 5-13.

Otro punto a considerar es que no todas estas comprobaciones son realizadas por los mismos agentes, si nos fijamos en las figuras 5-13, 5-14 y 5-15, podemos ver que las líneas que relacionan los milestones son de distintos colores, cada color indica un agente distinto, en la tabla 5-3 encontraremos el significado de cada color:

Color	*Agente que realiza la comprobación*
Rojo	CFMU
Negro	ATC
Azul	AO
Naranja	GH
Verde	Airport
Relaciones 1, 2 y 21	SimulatorInput

Tabla 5-3: Leyenda de agentes que actuan en la propagación de retrasos

Caso General:

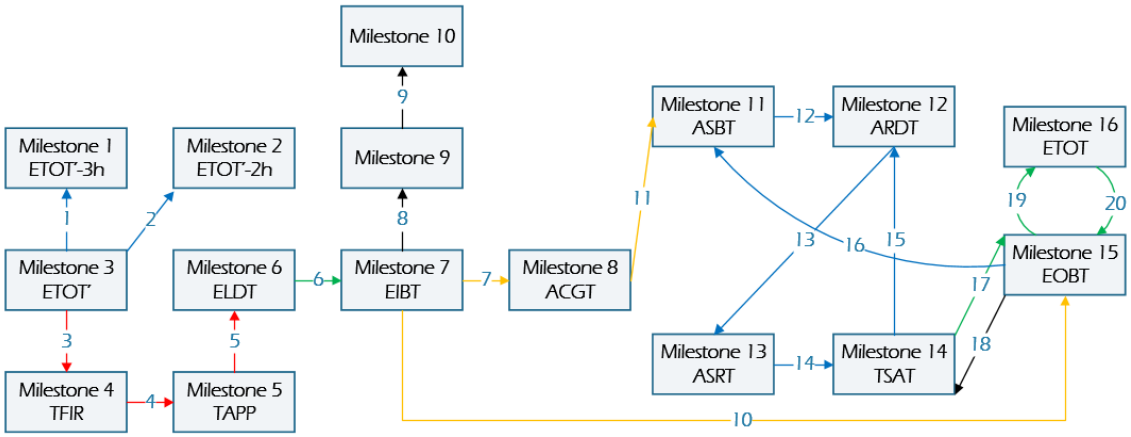


Figura 5-13: Propagación de retrasos caso general

En la figura 5-13 cada línea representa una comprobación que realiza uno de los agentes. Entre los milestones que se encuentran en los extremos de cada línea se comprueba que la diferencia de tiempo llegue a siempre a un mínimo. De manera que si el EOBT se viera retrasado, afectaría instantáneamente al ETOT, provocando que el avión despegue más tarde de lo previsto.

El programa está preparado para no permitir ninguna variación en un milestones una vez este ya ha sido completado. Por este motivo si después del ASBT (inicio boarding) tenemos un retraso en ETOT de 3 horas, la hora de ASBT no se verá modificada, pues ya se ha superado este milestone.

A continuación podemos encontrar la tabla 5-4 que nos muestra cual es la variable utilizada para realizar las relaciones entre los tiempos de milestone en cada caso:

Relación número:	Relación entre milestones:
1	tiempoMil1
2	tiempoMil2
3	Tiempo Vuelo
4	Tiempo en FIR
5	Tiempo en APP
6	Taxi-In
7	EIBTtoACGT
8	tiempoMil9
9	tiempoMil10

10	Turn around
11	ACGTtoASBT
12	ASBTtoARDT
13	ARDTtoASRT
14	ASRTtoTSAT
15	TSATtoARDT
16	EOBTtoASBT
17	TSATtoEOBT
18	EOBTtoTSAT
19	Taxi-Out
20	Taxi-Out

Tabla 5-4: Variables que definen el tiempo mínimo entre milestones

Solo aterrizan:
Por otra parte la tabla de comprobaciones se puede ver mínimamente modificada con los aviones que solo salen y los que solo llegan. En la figura 5-14 podemos ver gráficamente las comprobaciones que se dan cuando un avión solo llega. Como podemos observar es exactamente igual al caso general, pero eliminando todos los milestones pasados el 8.

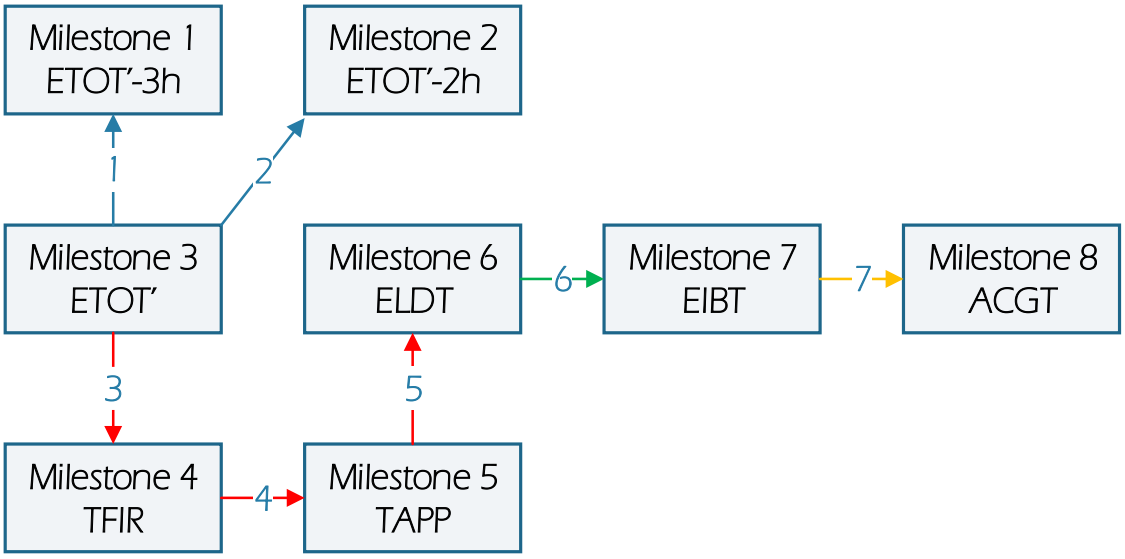


Figura 5-14: Propagación de retrasos caso solo aterrizan

Solo despegan:
En el caso de los aviones que solo salen de nuestro aeropuerto el efecto es el opuesto, solo tenemos las relaciones entre los milestones empezando por el milestone 10 tal y como podemos observar en la figura 5-15:

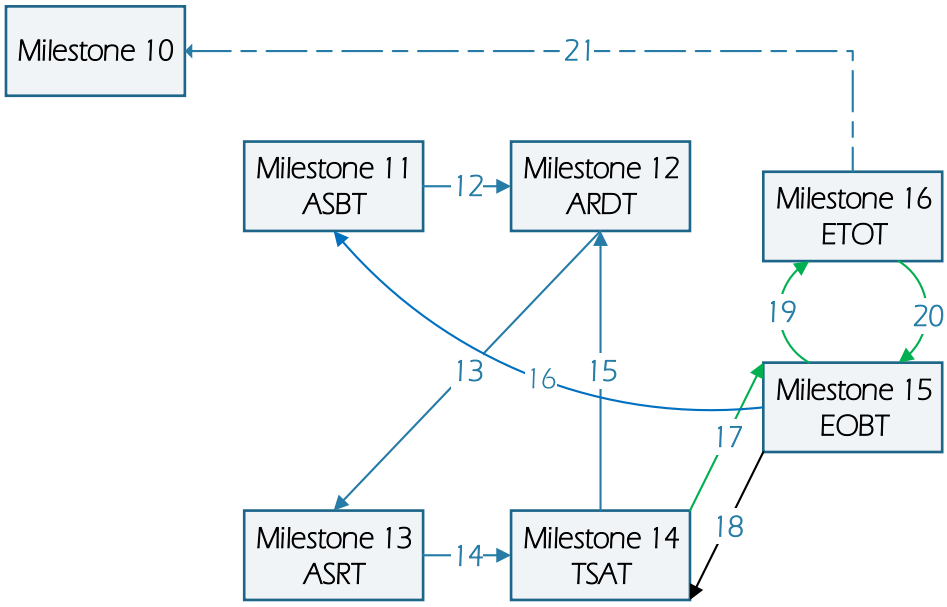


Figura 5-15: Propagación de retrasos caso solo despegan

En este caso se ha creado una nueva relación, la 21, entre el ETOT y el milestone 10, forzando al avión a tener siempre un mínimo de 3h entre el milestone 10 y el 16. De esta manera el avión aparece en el programa 3 horas antes del ETOT.
Como podemos ver en estos dos casos, solo salidas y solo llegadas, la cantidad de comprobaciones que realizamos es menor, pues muchos de los tiempos no se definirán, pues el avión no realiza el proceso completo dentro de nuestro aeropuerto.

Ahora ya sabemos todas las comprobaciones que realizara el simulador constantemente y quien las realiza, pero podemos observar dos tipos de comprobaciones distintas:

- **Comprobación con un milestone posterior**: Por ejemplo en la relación 17, se comprueba el milestone 14 con el 15. Cuando esto ocurre retrasaremos siempre el milestone posterior al mínimo tiempo necesario para que se cumpla la diferencia de tiempo obligatoria entre ambos milestones
- **Comprobación con un milestone anterior**: Por ejemplo en la relación 18, se comprueba el milestone 15 con el 14. Cuando esto ocurre retrasaremos siempre el milestone anterior al máximo tiempo posible de manera que se siga manteniendo como mínimo la diferencia de tiempo obligatoria entre ambos milestones.

5.6 *Datos Utilizados en el programa*

A lo largo de la simulación el programa utiliza muchos datos internos como son el tiempo de taxi-in i taxi-out, el tiempo mínimo que necesita un avión para realizar el turn around por defecto, entre otros. Estos datos son necesarios en el programa para realizar la simulación. Pero a continuación se explicara por qué se han decidido unos valores y no otros. El principal objetivo en cualquier decisión tomada a la hora de escoger el valor de un parámetro es el de dar el máximo realismo posible a la simulación.

En algunos casos el valor del dato es el real, pues ese es el valor que utilizan en el aeropuerto, y en otros mediante documentos externos se han realizado una serie de suposiciones para decidir otros valores dando el máximo realismo posible a la simulación.

A continuación podemos encontrar una guía sobre cómo se ha decidido los parámetros utilizados a lo largo de la simulación:

5.6.1 Tiempos de TurnAround por Modelo y Aerolínea

El tiempo de turn around se puede encontrar de dos maneras, primero mediante el documento Turn-Around-Times.txt y si en el documento no existe ningún valor que permita al programa encontrar un tiempo de turn around de un modelo y aerolínea se definirá siempre el mismo valor por defecto en función de la categoría del avión estudiado.

Valores por defecto

El programa pondrá un valor por defecto de 35 minutos cuando no se encuentre un valor en el documento Turn-Around-Times.txt. Se ha escogido 35 minutos pues es el tiempo mínimo permitido de turn around para cualquier avión en el aeropuerto de Barcelona, no puede ser que un avión llegue i despegue en Barcelona en menos de 35 minutos. A este tiempo se ha decidido añadir un extra para los aviones más grandes, de manera que el tiempo mínimo por defecto de turn around finalmente es de:

Cat 4 → 35 min
Cat 3 → 40 min
Cat 2 → 45 min
Cat 1 → 50 min

El motivo de este aumento de tiempo es que como más grande es un avión más tiempo necesita para realizar el turn around. Entre otras cosas el tiempo para realizar el Ground Handling se verá aumentado, al igual que el tiempo necesario para hacer el boarding, pues podrá entrar más gente…

Los valores se han decidido para seguir un poco con el modelo de la tabla 5-5 perteneciente al trabajo de final de grado de la referencia [13].

Tipo de aeronave Actividad	L	M	H
Conexión con la pasarela	1	1	1
Desembarque de pasajeros	4	8	12
Descarga del equipaje	7	11	22
Carga del equipaje	10	18	26
Agua potable	5	6	17
Aguas residuales	6	10	21
Combustible	7	13	29
Limpieza de la cabina	7	13	32
Catering	10	15	32
Embarque de pasajeros	10	15	20
Desconexión con la pasarela	1	1	1
"*push-back*"	2	2	2
TOTAL	28	42	68

Tabla 5-5: Tabla con los tiempos previstos para las actividades en gate [13]

El programa utiliza unos valores inferiores, pues queremos plasmar el caso en el que todo sucediera con el mínimo tiempo posible, ya que el valor del tiempo de Turn Around hace referencia al tiempo mínimo necesario en lugar del estándar.

Valores documento Turn-Around-Times.txt

Los valores del documento Turn-Around-Times.txt son valores seleccionados alearotiamente teniendo en cuenta la categoría de cada avión. Se han decidido unos valores para cada uno de los modelos. De la misma manera, en algunos casos, se ha decidido que unas aerolíneas tardaran más que otras en realizar el turn around de un mismo modelo. De manera que es posible simular que un avión de Ryanair tarde menos en realizar el turn around de un avión del mismo modelo pero de otra aerolínea

5.6.2 Aerolíneas utilizadas en el ejemplo

El programa tiene una serie de aerolíneas con agente propio mientras que todas las demás trabajan mediante el agente AO-GEN. Recordamos que la lista de aerolíneas con agente propio puede ser modificada en cualquier momento siguiendo las instrucciones del apéndice “Manual del Software”. En el ejemplo de utilización del programa podemos ver que viene con unas aerolíneas con agente propio como son:

```
RYR IBE VLG AEA EZY DLH NAX AFR
```

Estas aerolíneas han sido seleccionadas pues son las que más operaciones realizan en el aeropuerto de Barcelona.

5.6.3 Empresas de Ground Handling

El programa tiene una serie de empresas de ground handling con agente propio (GH) mientras que todas las demás trabajan mediante el agente GH-GEN. Cada empresa tiene una serie de contratos que definirán las aerolíneas a las que prestan servicios cada una de las empresas, las aerolíneas que no realicen el ground handling con una de las empresas con agente propio serán simuladas desde el GH-GEN. Recordamos que la lista de ground handling con agente

propio y sus contratos pueden ser modificados en cualquier momento siguiendo las instrucciones del apéndice “Manual del Software”.
En el ejemplo de utilización del programa tenemos definidas las siguientes empresas de GH con sus correspondientes contratos:

```
IBEServices-IBE Lesma-RYR FrankfurtAGS-TAP-DUB-VLG
    AviaPartner-AEA-NAX-AFR-MON Swissport-SWR-WZZ.
```

Las empresas de FrankfurtAGS, AviaPartner y Swissport han sido seleccionadas pues son de las que más operaciones realizan a nivel Europeo, pero los contratos son inventados únicamente con el objetivo de probar el buen funcionamiento del programa.

5.6.4 Categorías de los aviones

La categoría de los aviones es un parámetro que se define según los valores que entran en el programa mediante el documento Airport.txt el cual contiene una lista de aviones con la categoría correspondiente a cada modelo según las especificaciones ICAO. La categoría de cada avión se ha obtenido de las referencias [14], [15] y [16].

Para los modelos que no aparecen en la lista el simulador les asignara por defecto categoría = 4. El motivo es que se ha considerado que la lista inicial es suficientemente grande como para considerar que los aviones que no aparecen en ella son aviones pequeños que no aparecían en la lista de aviones utilizadas de las referencias.

5.6.5 Tiempos para pre-departure Sequence

El valor de la capacidad de operaciones por hora de un aeropuerto es una de las dadas más importantes. En este simulador no se ha considerado los casos de tener congestión en ATM, en taxi ways ni en la gate. Por eso el cuello de botella de nuestro aeropuerto que definirá la capacidad del aeropuerto vendrá definido por la capacidad máxima de la pista. El simulador considera que trabajamos en un aeropuerto con dos pistas activas, una solo para despegues y la otra destinada a aterrizajes. Por lo que la capacidad de la pista y del aeropuerto vendrá dada por como de seguidos pueden llegar o salir los aviones de nuestro aeropuerto.

Para decidir el tiempo mínimo entre dos aviones a la hora de llegar o salir hemos utilizado la tabla 5-6 extraída de la referencia [16]. En la tabla, las casillas vacías se consideran 3NM entre los 2 aviones.

		Follower			
		A380	Heavy	Medium	Light
Leader	A380		6NM	7NM	8NM
	Heavy		4NM	5NM	6NM
	Medium				5NM
	Light				

Tabla 5-6: Tabla de distancias mínimas entre aviones

Esta tabla indica la distancia mínima entre dos aviones al llegar y salir de un aeropuerto según ICAO. Aunque esta tabla nos da una idea de cómo tiene que ser la separación entre aviones, no la podemos aplicar directamente en el simulador, pues nosotros trabajamos con el tiempo entre aviones, no con la distancia. Hemos considerado que la velocidad de rotación de un avión está alrededor de los 140NM/h (2.3NM/min). Este valor se ha decidido mirando la [17], donde se da ese tiempo de rotación para un avión considerablemente grande (47000Kg), por lo que se trata de un avión de categoría 2 [18]. Por este motivo se ha considerado esta velocidad de rotación válida para todos los aviones. Dándonos lugar a la tabla 5-7, definiendo la separación entre aviones en tiempo en lugar de distancia, esta separación será válida para aterrizajes y despegues.

		Persiguiendo			
		Cat 1	**Cat 2**	**Cat 3**	**Cat 4**
Líder	**Cat 1**	2 min	3 min	4 min	4 min
	Cat 2	2 min	2 min	3 min	3 min
	Cat 3	2 min	2 min	2 min	3 min
	Cat 4	2 min	2 min	2 min	2 min

Tabla 5-7: Tabla con los tiempos mínimos entre aviones

Los valores de la tabla 5-7 son los que tiene el programa por defecto para marcar el tiempo mínimo entre dos aviones según la categoría de estos. Una vez tenemos esta tabla consideraremos que la máxima capacidad de pista es la máxima capacidad de operaciones que puede realizar el aeropuerto. Si tenemos 2 pistas y cada una realiza una operación cada 2 minutos como máximo, tendremos que la máxima capacidad del aeropuerto es de 60 operaciones/h, 30 de llegada y 30 de salida.

5.6.6 Constantes cálculo tiempos entre Milestones

Como ya se ha visto, un avión realiza un total de 16 milestones, cada milestone significa una acción realizada por el avión. Algunos de los tiempos de milestone vienen dados con los documentos de entrada, aunque pueden ser modificados durante la simulación, y los otros son calculados por el programa durante la simulación.

Para calcular estos tiempos de milestone se utilizan tiempos como los de taxi, turn around o valores de constantes. Recordamos que el programa funciona de manera que cada vez que se actualiza un valor de un tiempo este afecta a otros creando una malla que relaciona todos los milestones entre ellos. Podemos observar todas las relaciones que comprueba el programa en la figura 5-16:

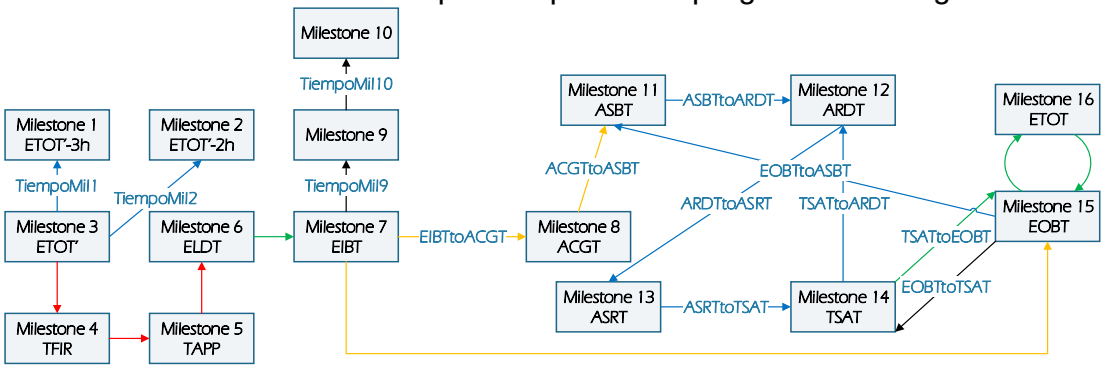


Figura 5-16: Propagación de retrasos caso general

En la figura 5-16 podemos observar cuales son los milestones que tienen un tiempo mínimo entre ellos. Estas comprobaciones son las que hace el programa constantemente para todos los aviones durante la simulación. A continuación, en la tabla 5-8, explicaremos por qué se ha decidido cada valor:

Constante	*Tiempo mínimo permitido (min)*	*Motivo*	*Referencia*
tiempoMil1	180	Tiempo definido por ICAO	ICAO
tiempoMil2	120	Tiempo definido por ICAO	ICAO
tiempoMil9	3	tiempo para Confirmación de la EOBT	-
tiempoMil10	5	Tiempo para definición de TSAT	-
EIBTtoACGT	3	Tiempo entre llegar a gate y empezar el handling	Documento máster ITAérea
ACGTtoASBT	1	Tiempo mínimo para empeza con el boarding una vez empezado el handling	-
ASBTtoARDT	20	Tiempo que se tarda en realizar todo el boarding	
ARDTtoASRT	1	Tiempo entre que el avión está preparado para salir y los pilotos piden permiso para salir	-
ASRTtoTSAT	1	Tiempo que tardan a dar permiso al avión una vez lo ha pedido	-
TSATtoEOBT	5	Tiempo de encender motores e iniciar taxi	Documento máster ITAérea
EOBTtoTSAT	5	Tiempo de encender motores e iniciar taxi	Documento máster ITAérea
TSATtoARDT	2	Tiempo entre que el avión está preparado y se le da permiso para salir	-
EOBTtoASBT	27	Tiempo mínimo entre inicio embarque i salida	-

Tabla 5-8: Información sobre las relaciones entre milestones

De la tabla 5-8, vemos que algunos valores que no tienen ninguna referencia, eso es porque estos valores han sido escogidos poniendo el tiempo que este podía necesitar en realizar cada tarea, pero no tienen ninguna fuente que asegure que ese valor es el que realmente necesitaríamoos en un aeropuerto

Volviendo a la figura 5-16, también podemos ver que alguna de las relaciones no tienen ninguna constante con la que se relacionan. El motivo es que algunas de estas relaciones vienen definidas por el usuario en la interfaz:

-Tiempo Taxi in: Relación entre milestone 6 y milestone 7.

-Tiempo Taxi Out: Relación entre milestone 8 y milestone 16.

Las relaciones entre milestone3→milestone4→milestone5→milestone6 tampoco tienen ninguna constante entre ellas, pues cada avión tendrá unos tiempos distintos en función de la ruta realizada por el avión para llegar a nuestro aeropuerto.

5.6.7 Taxi-in/Taxi-out

Los tiempos de taxi que utiliza el programa en nuestra versión son de 15 minutos para el taxi-in y 20 minutos para el taxi-out, son valores inventados, pues cada aeropuerto tiene un valor propio de tiempo de taxi según como estén diseñadas las pistas de cada aeropuerto, de manera que a pesar de venir el programa con unos valores por defecto, estos pueden ser cambiados para simular los tiempos de taxi de un aeropuerto en concreto con tal de poder estudiarlo con el máximo realismo posible.

CAPITULO 6. Resultados y Futuras líneas de mejora

6.1 *Ejemplo Utilidades del programa: Análisis resultados*

Tal y como se ha comentado ya en varias ocasiones el objetivo de este proyecto no es el de analizar el tráfico aéreo en un aeropuerto para poder mejorar el flujo de tráfico aéreo. Sino que el objetivo es el de diseñar y crear un programa capaz de simular el A-CDM de un aeropuerto y que a la vez este nos permita realizar un análisis de los resultados con tal de poder ver cómo serán los retrasos y como se desenvuelven según varían algunos de los parámetros del aeropuerto.

Antes de mostrar los resultados encontrados queremos comentar que el programa utiliza la librería jmathplot.jar. Esta librería ya viene instalada en el programa Eclipse al realizar la instalación del programa. Si por algún motivo no dispones de la librería la puedes encontrar en la referencia [19].

A continuación vamos a mostrar un ejemplo de los resultados y los análisis que podría obtener un usuario del simulador, se ha decidido utilizar la muestra del tráfico aéreo del 4 de diciembre del 2014 en Europa que se ha extraído de la página web de Eurocontrol tal y como se explica en el apéndice “Obtención Datos DDR2 Eurocontrol”. Estos datos contienen información de todos los vuelos Europeos entre las 0030 y las 2300.

Para poder realizar un análisis de resultados más interesantes, se ha repetido el análisis del día 4 de diciembre de 2014 variando ciertos parámetros de la simulación para ver cómo estos afectaban a los resultados finales.

A continuación observamos una tabla con los parámetros seleccionados para cada simulación realizada:

Tal y como se puede observar en la tabla 6-1, se han realizad 3 simulaciones de los mismos datos, realizando pequeñas variaciones entre ellas:

PARÁMETRO	**ANÁLISIS A**	**ANÁLISIS B**	**ANÁLISIS C**
DÍA DE SIMULACIÓN	4/12/2014	4/12/2014	4/12/2014
AEROPUERTO DE ESTUDIO	LEBL	LEBL	LEBL
TAXI IN TIME	15	15	15
TAXI OUT TIME	20	20	20
VELOCIDAD SIMULACIÓN	1740	2250	2250
OP/H-IN	30	30	30
OP/H-OUT	30	30	30
RETRASOS TIPO 1:	NO	NO	NO
RETRASOS TIPO 2:	NO	NO	NO
RETRASOS TIPO 3	NO	NO	NO
RETRASOS TIPO 4:	NO	NO	**SI**
RETRASOS ALEATORIOS:	NO	**SI**	NO
POLÍTICA CONGESTIÓN:	1	1	1

Tabla 6-1: Parámetros de simulación utilizados

- **En el caso A**: no se ha considerado ningún retraso externo ni la posibilidad de retrasos aleatorios.
- **En el caso B**: no se ha considerado ningún retraso externo. Pero esta simulación incluirá retrasos aleatorios, el retraso que se aplicará a cada avión seguirá una distribución normal (σ^2=4), que se aplicará después de realizar las acciones de Taxi, tanto Taxi-In como en Taxi-Out.
- **En el caso C**: no se han considerado retrasos estadísticos, pero si una reducción de la capacidad de pista que afectará únicamente a la pista de salidas, bajando su capacidad a 20 operaciones/h en lugar de las 30 operaciones/h habituales. Esta reducción de la capacidad de pista será efectiva entre las 1200 i las 1500 i el simulador tendrá constancia de esta información a las 1000. Esta es la línea que se encuentra en el documento retrasos.txt: `4 1200 1500 1000 20 -`

Tras realizar estas 3 simulaciones hemos guardado los resultados de cada una para poder comprobarlos utilizando la función de análisis del simulador.

6.1.1 Análisis resultados caso de simulación con/sin retrasos aleatorios

Para realizar la comprobación de los resultados entre A y B nos hemos centrado en el estudio del parámetro ETOT, pues en este punto los aviones de la simulación B ya han realizado las dos carreras por las pistas de taxi del aeropuerto. Como es lógico el retraso aleatorio del Taxi-Out afectará directamente al tiempo ETOT, pero el retraso aleatorio en el Taxi-In (afecta directamente a EIBT), que inicialmente parece que no afecta al ETOT, podría terminar afectando también este tiempo debido a la propagación de retrasos que podría sufrir el avión.

A continuación, en la figura 6-1 mostramos los datos obtenidos al realizar el análisis de estas dos simulaciones estudiando ETOT:

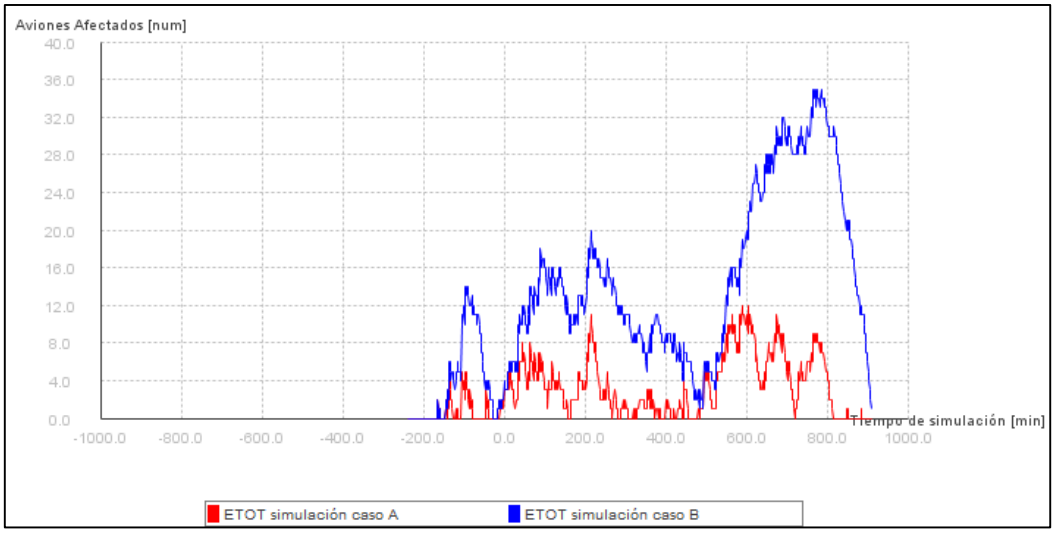


Figura 6-1: Gráfica resultados con/sin retrasos aleatorios

La gráfica de la figura 6-1 muestra la cantidad de aviones que tendrían de haber realizado el ETOT en cada instante de tiempo, pero debido a los retrasos aún no han podido realizar esa acción. La línea roja muestra los aviones de la simulación sin retrasos mientras que la azul muestra los aviones de la simulación con retrasos aleatorios. Como era de suponer la línea azul está por encima de la roja, pues los aviones en esa simulación han sufrido más retrasos de los necesarios.

La causa de que la gráfica azul llegue a distanciarse tanto de la roja en algunos momentos, es debido a que a pesar de que los retrasos que se aplican debido a la normal son solo de unos pocos minutos, esta variación puede causar que el avión este cambiando el slot en el que despega de la pre-departure sequence. Este hecho sumada a la decisión de realizar la simulación de PoliticaCongestion=1, provoca que cuando un avión mueve su slot, este se retrasa hasta el primer slot vacío de la pre-departure sequence.

En la figura 6-2 vemos una serie de datos relacionados con el análisis de los resultados de ambas simulaciones (caso A y caso B). Como podemos ver, el avión que sufre más Delay aumenta mucho entre ambas simulaciones de 60 minutos a 252, esto es básicamente debido a la política de congestión, pues ese avión ha recibido un pequeño retraso al realizar el Taxi, i ha tenido de retrasar mucho su ETOT debido a que no encontraba ningún slot vacío para realizar la salida del aeropuerto.

Datos Análisis:	Tabla 1	Tabla 2
Num.Aviones	339	339
Total Minutos Delay	3415	14993
Average Aviones/min	3.0	13.0
Average minutos delay/avión	10.0	44.0
Avión max. delay		
Avión	IBE0930	VLG1313
Delay	60	252
ETOT a las	1145	2127

Figura 6-2: Resultados análisis caso A (tabla 1) y caso B (tabla 2)

6.1.2 Análisis resultados caso de simulación con/sin reducción de la capacidad de pista

Para realizar la comprobación de los resultados entre A y C nos hemos centrado en el estudio del parámetro ETOT. El motivo de este estudio es que en la simulación C tenemos una reducción de la capacidad de pista de despegue, que entre las 12 y las 15 soportará un tráfico de 20 aviones por hora en lugar de los 30 nominales.

A continuación, en la figura 6-3 mostramos los datos obtenidos al realizar el análisis de estas dos simulaciones estudiando ETOT:

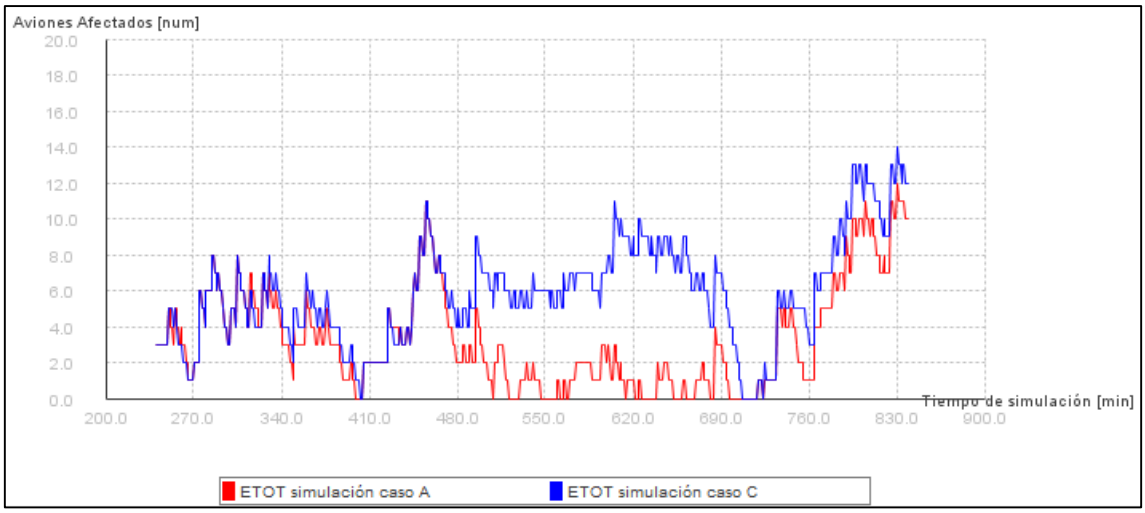


Figura 6-3: Gráfica resultados con/sin reducción capacidad de pista

La gráfica de la figura 6-3 muestra la cantidad de aviones que tendrían de haber realizado el ETOT en cada instante de tiempo, pero debido a los retrasos aún no han podido realizar esa acción. La línea roja muestra los aviones de la simulación sin retrasos mientras que la azul muestra los aviones de la simulación con la reducción de la capacidad de pista.

Podemos ver que al inicio las 2 gráficas van juntas, pues la pista del caso C aun funciona con su máxima capacidad, pero en un momento esta empieza a acumular aviones que esperan, pues la capacidad de la pista se ve reducida y los aviones tienen de esperar a que llegue su slot 3 horas después la capacidad de la pista de salidas vuelve a recuperar la capacidad nominal de 30 operaciones por hora y las 2 gráficas vuelven a ir juntas.

En la gráfica solo se muestra el período de tiempo comprendido entre las 0800 y las 1800, para ver más detalladamente el fragmento de tiempo que contiene la reducción de la capacidad de pista (1200-1500).

En la figura 6-4 vemos una serie de datos relacionados con el análisis de los resultados de ambas simulaciones (casi A y caso C). Podemos ver que en el segundo caso el avión con más retraso pasa a ser uno que quería salir dentro de la franja horaria donde tenemos la reducción de la capacidad de pista.

Datos Análisis:	Tabla 1	Tabla 2
Num.Aviones	163	183
Total Minutos Delay	1962	3509
Average Aviones/min	0.0	0.0
Average minutos delay/avión	0.0	0.0
Avión max. delay		
Avión	IBE0930	IFA836
Delay	60	178
ETOT a las	1145	1513

Figura 6-4: Resultados análisis caso A (tabla 1) y caso C (tabla 2)

Destacar también que las opciones *Average Aviones/min* y *Average minutos delay/avión* tienen un valor de 0 en los 2 casos. Esto es debido a que el programa devolverá estos dos parámetros como 0 al simular con una ventana de tiempo.

6.2 *Futuras líneas de mejora*

El programa simula el funcionamiento de un A-CDM en un aeropuerto. Pero se podrían añadir muchos parámetros, métodos y roles que darían al simulador un comportamiento más parecido al que encontramos en un aeropuerto real hoy en día. A continuación dejaremos plasmadas un par de ideas sobre donde se podrían centrar los esfuerzos con tal de seguir mejorando el programa, con tal de conseguir una simulación más realística o de conseguir darle más utilidad, dándole nuevos métodos para trabajar los resultados y poder realizar diferentes estudios de los que realiza actualmente el simulador.

6.2.1 Propuestas de continuación del programa

A continuación se expondrán una serie de ideas y propuestas que no han sido programadas. Pues al tratarse de un solo trabajo de fin de grado con una duración prevista de un cuatrimestre, no nos ha dado tiempo a implementar todas las funcionalidades que nos habría gustado. De todos modos, intentaremos dejar claro un par de ideas de cómo se tendría planeado continuar el proyecto en caso de tener tiempo suficiente.

- **Mejorar Pre-departure Sequence:** Añadir al programa un modelo de predeparture sequence más realista, actualmente tenemos dos modelos de predeparture sequence programados tal y como se explica en el apartado 4.6.3. El primero considera que cuando un avión intenta entrar en la predeparture sequence tendrá de buscar un slot donde situarse sin afectar a los otros aviones (todo el retraso lo sufre un avión), mientras que el segundo caso implementado considera que cuando un avión intenta entrar en la predeparture sequence moverá a todos los aviones que quieren salir más tarde que su tiempo de salida previsto (el retraso se reparte entre muchos aviones). Se podrían añadir nuevos modelos para realizar la predeparture sequence como considerar slots que pertenecen a las aerolíneas, y al retrasar un avión tendría que ser retrasado hasta el próximo slot vacío o al próximo slot de la compañía…
- **Aumentar la lista de Situaciones** por las que se pueden tener en cuenta muchos más casos para los retrasos de tipo 1 y tipo 2
- **Nuevas políticas y comportamientos:** Implementar nuevas políticas y comportamientos a los agentes por ejemplo, permitir que una aerolínea cancele un avión cuando este supera un límite de retraso.
- **Adelantar tiempos:** Permitir al programa adelantar tiempos en ciertas situaciones. Ahora mismo el programa solo acepta

modificaciones que retrasen un tiempo, nunca que lo adelanten, en una versión futura, se podría aplicar que el programa pudiera adelantar tiempos, para solventar situaciones que ahora no están permitidas como por ejemplo el caso de que un avión retrasado encuentre un slot vacío para despegar, pero ya no lo puede coger, pues eso significaría adelantar el ETOT.

- **Interfaz pedagógica:** También se podría implementar al programa una serie de interfaces y explicaciones de manera que se pudiese usar con objetivos pedagógicos para ayudar a los estudiantes a entender todo lo relacionado con el Airport-CDM

Con tal de mejorar el funcionamiento del simulador probablemente nos veremos en la obligación de implementar agentes externos encargados de realizar unos cálculos concretos. De manera que alguien le mandara un mensaje preguntándole un valor o acción a tomar y este nuevo agente le responderá con la solución o con un guion de actuación. A continuación encontraremos un par de ejemplos de agentes externos que podrían ser implementados en un futuro en el programa.

- **Agente encargado de "asignación de gates":** Consistiría en la creación de un agente encargado de decidir en qué gate estacionará cada avión en cada caso en función de la ocupación de las otras gates, de la categoría del avión y de la aerolínea.

  De manera que este agente será el encargado de asegurar que dos aviones no intentan estacionar en la misma gate, que si en el momento del aterrizaje de un avión con un gate determinado se encuentra que el que estaba antes va con retraso i aún no ha salido, tendrá de realizar una reasignación de las gate. A este agente también se le podría dar la capacidad de monitorizar constantemente el estado de nuestras gates, de manera que podamos saber en todo momento cuales están libres y cuáles no.
- **Agente encargado de calcular "Taxi Time":** Creación de un agente externo encargado únicamente de calcular el tiempo de taxi previsto para entradas y salidas. Este agente podría leer un documento de entrada que le diese información sobre los planos del aeropuerto, de manera que podrá saber dónde están localizado cada gate y la longitud de cada calle del Taxi y en qué sentido se circula en cada uno de ellos. De manera que podría calcular de una manera muy exacta el tiempo que tardará en realizar las rodaduras por pista cada uno de los aviones. Este agente podría incluso llegar a guardar la información de todos los aviones, de manera que podría saber que pistas estarán siendo utilizadas en cada momento y podría llegar a calcular con antelación los retrasos por congestión de pista.

6.2.2 Ejemplo Agente Externo

El agente Turn Around está creado como ejemplo para la creación de otros agentes externos.

La función de este agente es informar al GH del tiempo que tarda en realizar el turn around un avión determinado siempre que el GH le pregunte. En este apartado no nos centraremos en como decide cual es el tiempo de turn around de cualquier avión este agente, pues esto ya está explicado en el apartado 4.3.5.1. En este apartado nos centraremos en el proceso realizado con tal de conseguir que un agente externo funcione correctamente.

En primer lugar se tiene que decidir con que agente mantendrá comunicación en el caso del agente TurnAround, este mantendrá comunicación con el GH. Luego se ha escrito el código en el nuevo agente de manera que esté esperando en todo momento un mensaje del GH preguntando por un tiempo de TurnAround de un avión con un modelo en concreto, también se tiene que preparar el agente para responder dando una respuesta al agente que le ha mandado la pregunta. Es importante que en el mensaje de respuesta tengamos algún tipo de elemento que permita al agente que recibirá el mensaje distinguir ese mensaje de los otros que recibe normalmente, nosotros hemos definido un valor en el parámetro OpcionMensaje, el agente TurnAround siempre responderá al GH con el valor de OpcionMensaje = 8, que es el que indica que se trata de una información relacionada con el TurnAround.

De la misma manera se tiene que preparar al agente que solicitará la información, de manera que sepa a quien tiene que preguntar la información y a la vez, esté preparado para recibirla. En nuestro caso hemos cogido el agente GH, i se ha escrito una función, *AgenteExternoTurnAround*, que será la encargada de mandar un mensaje al agente externo TurnAround preguntándole por este tiempo en un caso concreto.

Justo después de mandar el mensaje se ha implementado un *blockingReceive* en el código, de manera que el programa queda a la espera de que el TurnAround responda, por este motivo se tiene que asegurar que el TurnAround mandará una respuesta en cualquier situación, pues sino la simulación quedaría parada aquí a la espera de un mensaje que nunca llegaría.

Al escribir este nuevo blockingReceive en el código del GH tenemos de considerar el caso en que mientras el agente GH espera respuesta del TurnAround reciba algún otro mensaje y active este blockingReceive antes de obtener la respuesta del TurnAround. Para solucionar esto se ha añadido al agente GH una ArrayList, *AvionesInicio*, que contendrá estos mensajes guardados hasta que se reciba la respuesta del TurnAround. De manera que cuando termine con los cálculos, antes de pararse de nuevo a la espera de mensajes, realizará los cálculos correspondientes a los mensajes que tenga guardados en esta ArrayList.

CAPITULO 7. CONCLUSIONES

Este proyecto se planteó para sentar las bases de lo que iba a ser un entorno de simulación basado en agentes para simular los procedimientos y el tráfico aéreo en un aeropuerto con Airport Collaborative Decision Making, siguiendo las directrices definidas por Eurocontrol en su manual de implementación A-CDM. Además, se decidió trabajar en una plataforma basada en agentes para así poder obtener las comunicaciones generadas durante la simulación, útiles para análisis posteriores, y poder ofrecer la posibilidad de un desarrollo independiente de los diferentes agentes del programa.

Una vez concluido el proyecto podemos asegurar que se han cumplido estos objetivos principales y, además, otros objetivos que se han ido marcando a lo largo del desarrollo del proyecto.

Estos objetivos adicionales se han cumplido al implementar diversas funcionalidades extra, como la posibilidad de utilizar tráfico aéreo real, de añadir retrasos y situaciones adversas en la simulación, dar la posibilidad de realizar un análisis de los resultados, y otras funcionalidades que ya se han comentado.

También se ha desarrollado el proyecto de manera que facilite que futuros TFG puedan nutrirse de este programa para trabajar en mejorarlo o, tal y como se ha preparado y gracias a haber utilizado un sistema multiagente, trabajar en módulos externos e independientes que mejoren el realismo de las simulaciones y se puedan añadir al programa fácilmente, gracias a los ejemplos de agentes externos y a haber simplificado las comunicaciones y los datos intercambiados.

En lo personal este proyecto nos ha servido para meternos más en un tema que ya de inicio nos parecía interesante, nos ha servido para ver la complejidad y la utilidad tanto de ACDM como de los diferentes procesos de ATM en general.

Además, nos ha ayudado a dar un paso más allá en lo que se refiere a la gestión de proyectos y al desarrollo de software. Este software se ha desarrollado siguiendo un ciclo de vida de ingeniería de software para asegurar la correcta realización de todos los pasos, y se ha tenido que organizar el trabajo para que ambos programadores trabajaran eficientemente y no se molestaran.

También nos ha permitido adquirir conocimientos y competencias más que interesantes en desarrollo de sistemas multiagente y programación en JADE y JAVA.

Por último comentar que la versión en papel de esta memoria contiene un CD con el código del programa y algunos ficheros de datos, tanto inputs como resultados, de ejemplo. En caso de no tener acceso a este CD, es posible acceder al código a través de este link de Github:

https://github.com/A-CDMteam/SimuladorV2.0

BIBLIOGRAFIA

[1] Airport CDM Applications Guide, EUROCONTROL, July 2003

[2] Airports Council International, Eurocontrol, IATA, "Airport CDM implementation. The Manual". Tech. Rep. Ed. 1.4. April 2008.

[3] Ivan García, Miguel Valero, Xavier Prats, Luís Delgado, “Agent-based simulation framework for airport collaborative decision making”. May 2014

[4] EUROCONTROL, Aiport CDM Implementation Manual, Edition 4, March 2012.

[5] Eurocontrol, "Airport CDM Cost Benefit Analysis," Tech. Rep. Version 4. April 2012.

[6] Notícia nuevo aeropuerto CDM, http://www.eurocontrol.int/news/venice-becomes-3rd-cdm-italian-airport, 23 enero 2015

[7] Alejandro Alonso, “Implementación del algoritmo Computer Assisted Slot Allocation (CASA) de la Central Flow Management Unit (CFMU)”, junio 2011

[8] Juan Fco. Garamendi, “Agentes Inteligentes: JADE”, Abril 2004

[9] Web programación JADE, http://programacionjade.wikispaces.com, enero 2015

[10] Giovanni Caire, TILAB, “JADE TUTORIAL, JADE programming for beginners”, June 2009

[11] Michael Wooldridge, Nicholas Jennings, David Kinny, “The Gaia Methodology for Agent-Oriented Analysis and Design”

[12] José M. Drake, Patricia López, CTR, ”Ingeniería Software Verificación y Validación”. 2011

[13] Alejando Agustí Chávez, Juan José Ramos González, UAB, “Gestión de Recursos de Handling en Aeropuertos Congestionados”, Junio 2013

[14] Web de Nats, Performance through Innovation, http://nats.aero/blog/wp-content/uploads/2013/06/HoldingInfographic.pdf, Enero 2015

[15] Web http://www.flugzeuginfo.net/table_accodes_en.php#0, Enero 2015

[16] Presentación on Wake Turbulence Re-Categorization Phase I Methodology and Safety Case, TU Berlin, 20 Junio 2011

[17] Web http://www.jaon.es/despegue/despegue.htm, Enero 2015

[18] Dr. Antonio A. Trani, Department of Civil Engineering Virginia Tech, “Aircraft Classifications”, Enero 2014

[19] Web programación Jmathplot https://code.google.com/p/jmathplot/

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

APÉNDICES

TÍTULO DEL TFG: Entorno de simulación basado en agentes para A-CDM (Airport Collaborative Decision Making)

TITULACIÓN: Grado en Ingenieria de Aeronavegación

AUTORES: Ivan Garcia Vasco
Xavier Morell Llorens

DIRECTOR: Miguel Valero Garcia
Xavier Prats Menéndez
Luís Delgado Muñoz

FECHA: 6 de febrero del 2015

ÍNDICE APÉNDICES

APÉNDICE A. MILESTONE APPROACH EXTENDIDA

A.1 Introducción

Tal y cómo se ha explicado en el Capítulo 2 de la memoria, Eurocontrol define una serie de eventos o *Milestones* a través de los cuales divide el ciclo CDM de un avión (desde el despegue en el aeropuerto de origen hasta que sale de nuestro aeropuerto).

Tal y cómo dicta Eurocontrol en su manual [4], se han definido 16 milestones básicos. En la figura A-1 podemos ver estos milestones y la posición que ocupan en el desarrollo de un vuelo de un avión:

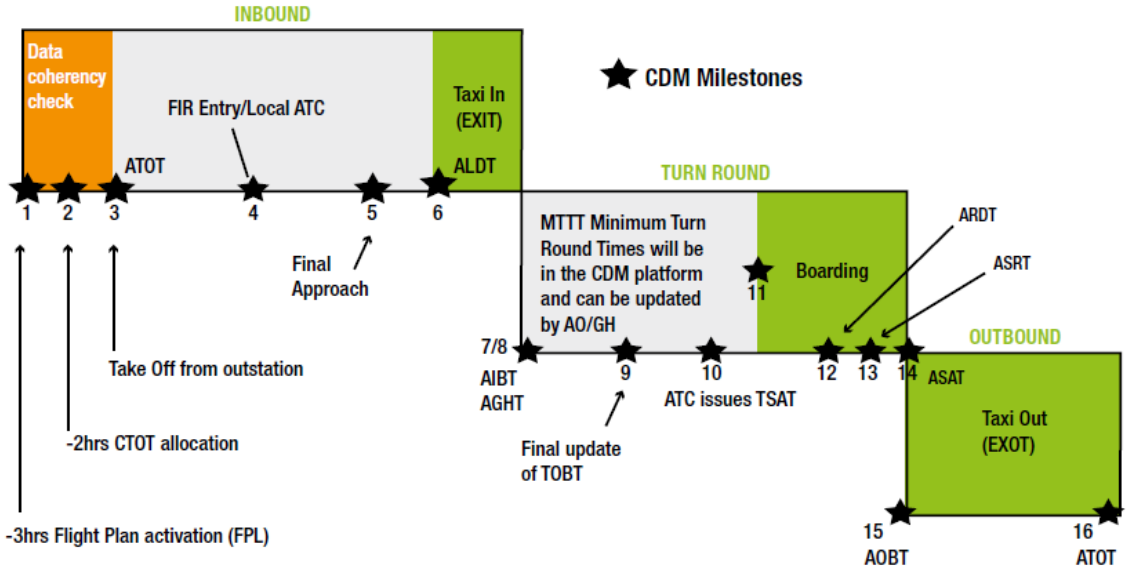


Figura A-1: Esquema Milestones [4i]

A continuación se explicará en que consiste cada uno de estos milestones, y que procedimientos principales lleva a cabo el programa en cada uno de ellos.

En cada uno de los milestones se comentará:

- Defincón breve de en que consiste el milestone (titulo del milestone).
- Tiempo en el que ocurre, en relación a los tiempos del avión.
- Status del avión durante ese milestone. Describe la situación en la que se encuentra el avión y sirve para poder entender más rápidamente la situación del avión una vez estamos mirándolo en la interfaz del Information Sharing
- Evento Trigger. Evento que hace que se inicie este milestone, situación del vuelo a la que llega el avión y hace que pase a considerarse que el avión esta en un milestone determinado.
- Agente que detecta este evento y avisa a los demás a través del Information Sharing.
- Tiempos que se definen en el ACDM por primera vez

- Tiempos que se actualizan en el ACDM
- Procesos generales que realizan los agentes en este milestone.
- Comentarios sobre el milestone

Esta explicación se ha realizado siguiendo la premisa de un vuelo sin problemas ni retrasos. Por lo tanto, veremos cómo todos los tiempos se definen y se calculan en el milestone 1, y en los siguientes milestones solo va cambiando el status del avión y los tiempos van pasando de *Estimated* a *Actual*. En una simulación real esto no suele ocurrir, ya que se detectan retrasos y se propagan entre aviones constantemente, y encontraremos que los agentes realizan cálculos en todos los milestones.

Como se comenta durante el análisis, los cálculos de los tiempos se pueden ver mejor en los apartados 4.3 y 5.5 de la memoria.

A.2 Milestone Approach

A.2.1 Milestone 1

Definición: Activación Flight Plan / Inicio CDM

Tiempo: 3horas antes del despegue en aeropuerto de origen

Status avión: INITIATED

Evento Trigger: Se publica el flight Plan (en caso de vuelo regular el flight plan ya estaba publicado, en este milestone solo se activa el FP y se inicia el CDM)
Agente: CFMU

Tiempos definidos: A partir de los datos que provienen del Flight Plan, el programa calcula todos los tiempos del avión, tal y cómo se puede ver en el apartado 5.5 de la memoria.
Estos tiempos hacen referencia a los eventos en los que ocurren los miletones, a continuación los vemos numerados, en orden temporal.
Tiempos: ETOT', TFIR, TAPP, ELDT, EIBT, ACGT, ASBT, ARDT, ASRT, TSAT, EOBT, ETOT.

Tiempos actualizados:

Procesos Agentes:
Todos los agentes actúan en el cálculo de los tiempos del avión. A continuación solo comentamos que tiempos calcula cada agente, para ver el orden de los cálculos y las relaciones entre los tiempos recomendamos pasar por el apartado 5.5 de la memoria.

CFMU	Publica el Flight Plan del avión (Inicio Milestone)

	Define por primera vez el avión en el CDM. Crea un avión nuevo y le añade toda la información relacionada con este, cómo: Callsign, Aerolínea, ADEP, ADE, Modelo, etc. Añade ELDT, ETOT', TFIR y TAPP al Information Sharing
ATC	Calcula TSAT
AIRPORT	Calcula EIBT y EOBT
Aircraft Operator	Calcula ARDT, ASRT y añade ETOT
Ground Handling	Calcula ACGT y ASBT

Comentarios

- La variable ETOT' se refieren al tiempo de Take-off en el aeropuerto de origen. Es un tiempo útil para aproximar tiempos de llegada a nuestro aeropuerto, pero el CDM de nuestro aeropuerto no puede cambiar.
  En un principio se extrae del FP, aunque se puede actualizar si el aeropuerto de origen tiene un CDM implementado o si ATC decide definir un GDP (Ground Delay Program).
- Las variables EOBT y ETOT también se pueden entender cómo variables "objetivo" (target), ya que son los tiempos que la aerolínea quiere cumplir para que el avión pueda seguir con sus demás vuelos del día.

Esto no nos afecta, ya que en todo momento el CDM busca un aeropuerto dinámico y rápido operando. Por tanto, consideramos que el tiempo estimado es el objetivo, ya que es el tiempo mínimo en hacer el turn-round de la aeronave.

- Milestone solo para iniciar el CDM y que los agentes verifiquen la información.

A.2.2 Milestone 2

Definición: Actualización de datos

Tiempo: 2 horas antes del despegue en aeropuerto de origen

Status avión: INITIATED

Evento Trigger: Actualización de variables
Agente: ATC

Variables Iniciadas: CTOT' (si hay regulación)

Variables Actualizadas: ELDT, EOBT (si hay regulación)

Procesos Agentes:

Los agentes solo actuarán en caso de que haya alguna regulación o alguna actualización de tiempos (algún retraso), en caso de no haberlo los agentes solo actualizarán las variables que toque.

Para ver cómo los agentes actualizan los tiempos y propagan un retraso, recomendamos visitar el apartado 4.3 y el 5.5, donde se puede ver que

agente actualiza cada tiempo y que relaciones hay entre los diferentes tiempos

CFMU	Actualiza los datos del avión si es necesario, 2h antes de ETOT' (Inicio Milestone)
ATC	
AIRPORT	
Aircraft Operator	
Ground Handling	

Comentarios
- Simplemente se actualiza la información, para mejorar su calidad.

A.2.3 Milestone 3

Definición: Despegue en aeropuerto de origen

Tiempo: ETOT'

Status avión: AIRBORNE

Evento Trigger: Despegue del avión en aeropuerto de origen
Agente: ATC aeropuerto de origen

Variables Iniciadas: ETOT' pasa a ser ATOT' (de Estimated a Actual)

Variables Actualizadas:

Procesos Agentes:

Los agentes solo actuarán en caso de que haya alguna regulación o alguna actualización de tiempos (algún retraso), en caso de no haberlo los agentes solo actualizarán las variables que toque.
Para ver cómo los agentes actualizan los tiempos y propagan un retraso, recomendamos visitar el apartado 4.3 y el 5.5, donde se puede ver que agente actualiza cada tiempo y que relaciones hay entre los diferentes tiempos

CFMU	
ATC	Detecta que el avión ha despegado de su aeropuerto de origen y actualiza status y tiempos (Inicio Milestone)
AIRPORT	
Aircraft Operator	
Ground Handling	

Comentarios

- A partir de este milestone el avión empieza a seguir el plan de vuelo que se encuentra en el IS. Por tanto, nos interesan y podemos manipular todas las variables de tiempo, que se irán actualizando durante el vuelo.

A.2.4 Milestone 4

Definición: Actualización de radar.

Tiempo: TFIR

Status avión: FIR

Evento Trigger: El avión entra en la FIR de nuestro aeropuerto

Agente: ATC

Variables Iniciadas:

Variables Actualizadas:

Procesos Agentes:

Los agentes solo actuarán en caso de que haya alguna regulación o alguna actualización de tiempos (algún retraso), en caso de no haberlo los agentes solo actualizarán las variables que toque.

Para ver cómo los agentes actualizan los tiempos y propagan un retraso, recomendamos visitar el apartado 4.3 y el 5.5, donde se puede ver que agente actualiza cada tiempo y que relaciones hay entre los diferentes tiempos

CFMU	
ATC	Detecta que el avión ha entrado en la FIR y actualiza status y tiempos (Inicio Milestone)
AIRPORT	
Aircraft Operator	
Ground Handling	

Comentarios

- Se vuelven a actualizar las variables de tiempo con datos más precisos, ya que se está reduciendo la incertidumbre que genera el EET (Estimated Elapsed Time)

A.2.5 Milestone 5

Definición: Actualización de radar

Tiempo: TAPP

Status avión: FINAL

Evento Trigger: El avión empieza la aproximación final
Agente: ATC

Variables Iniciadas:

Variables Actualizadas:

Procesos Agentes:

Los agentes solo actuarán en caso de que haya alguna regulación o alguna actualización de tiempos (algún retraso), en caso de no haberlo los agentes solo actualizarán las variables que toque.
Para ver cómo los agentes actualizan los tiempos y propagan un retraso, recomendamos visitar el apartado 4.3 y el 5.5, donde se puede ver que agente actualiza cada tiempo y que relaciones hay entre los diferentes tiempos

CFMU	
ATC	Detecta que el avión ha iniciado la aproximación final y actualiza status y tiempos (Inicio Milestone)
AIRPORT	
Aircraft Operator	
Ground Handling	

Comentarios
- Actualización de tiempos para reducir incertidumbre.

A.2.6 Milestone 6

Definición: Aterrizaje del avión

Tiempo: ELDT

Status avión: LANDED

Evento Trigger: Aterrizaje avión
Agente: ATC

Variables Iniciadas: ELDT pasa a ser ALDT (de Estimated a Actual)

Variables Actualizadas:

Procesos Agentes:

Los agentes solo actuarán en caso de que haya alguna regulación o alguna actualización de tiempos (algún retraso), en caso de no haberlo los agentes solo actualizarán las variables que toque.

Para ver cómo los agentes actualizan los tiempos y propagan un retraso, recomendamos visitar el apartado 4.3 y el 5.5, donde se puede ver que agente actualiza cada tiempo y que relaciones hay entre los diferentes tiempos

CFMU	
ATC	Detecta que el avión ha aterrizado y actualiza status y tiempos (Inicio Milestone)
AIRPORT	
Aircraft Operator	
Ground Handling	

Comentarios
- Actualización de tiempos

A.2.7 Milestone 7

Definición: IN-BLOCK

Tiempo: EIBT

Status avión: INBLOCK

Evento Trigger: Avión llega a Gate
Agente: ATC

Variables Iniciadas: EIBT pasa a ser AIBT (de Estimated a Actual)

Variables Actualizadas:

Procesos Agentes:

Los agentes solo actuarán en caso de que haya alguna regulación o alguna actualización de tiempos (algún retraso), en caso de no haberlo los agentes solo actualizarán las variables que toque.

Para ver cómo los agentes actualizan los tiempos y propagan un retraso, recomendamos visitar el apartado 4.3 y el 5.5, donde se puede ver que agente actualiza cada tiempo y que relaciones hay entre los diferentes tiempos

CFMU	
ATC	Detecta que el avión ha iniciado llegado a Gate y actualiza status y tiempos (Inicio Milestone)
AIRPORT	
Aircraft Operator	
Ground Handling	

Comentarios
- Actualización de tiempos

A.2.8 Milestone 8

Definición: Ground Handling Started

Tiempo: ACGT
Status avión: INLBOCK

Evento Trigger: Empieza el handling
Agente: GH

Variables Iniciadas: ACGT pasa a ser “Actual”

Variables Actualizadas:

Procesos Agentes:

Los agentes solo actuarán en caso de que haya alguna regulación o alguna actualización de tiempos (algún retraso), en caso de no haberlo los agentes solo actualizarán las variables que toque.
Para ver cómo los agentes actualizan los tiempos y propagan un retraso, recomendamos visitar el apartado 4.3 y el 5.5, donde se puede ver que agente actualiza cada tiempo y que relaciones hay entre los diferentes tiempos

CFMU	
ATC	
AIRPORT	
Aircraft Operator	
Ground Handling	Inicia handling al avión y actualiza tiempos (Inicio Milestone)

Comentarios
- Llegados a este punto hay dos posibilidades: Que el avión tenga que volver a cargar pasajeros y salir, o que dicho avión tenga que esperar antes de volver a cargar pasajeros y realizar otro vuelo.

- A partir de este Milestone el avión cambia de FP, siendo ahora nuestro aeropuerto el aeropuerto de origen.

A.2.9 Milestone 9

Definición: Confirmación Final del TOBT

Tiempo: Antes del EOBT (tiempo predefinido en el programa, depende del aeropuerto)

Status avión: INBLOCK

Evento Trigger: Definición TOBT definitivo
Agente: ATC

Variables Iniciadas: EOBT pasa a ser TOBT (de Estimated a Target)

Variables Actualizadas:

Procesos Agentes:

Los agentes solo actuarán en caso de que haya alguna regulación o alguna actualización de tiempos (algún retraso), en caso de no haberlo los agentes solo actualizarán las variables que toque.

Para ver cómo los agentes actualizan los tiempos y propagan un retraso, recomendamos visitar el apartado 4.3 y el 5.5, donde se puede ver que agente actualiza cada tiempo y que relaciones hay entre los diferentes tiempos

CFMU	
ATC	Realiza predeparture sequence y a partir del EOBT del avión y el tráfico en el aeropuerto define el TOBT; actualiza tiempos si es necesario (Inicio Milestone)
AIRPORT	
Aircraft Operator	
Ground Handling	

Comentarios

- Si no hay problemas de congestión y se ha realizado una planificación correcta, no debería haber diferencia entre el EOBT y el TOBT.
- La calidad de un ACDM se basa en cuan preciso es en calcular un TOBT que se ajuste con el real.

A.2.10 Milestone 10

Definición: Definición TSAT (Target StartUp Aproval Time)

Tiempo: Antes del EOBT (tiempo predefinido en el programa, depende del aeropuerto)

Status avión: SEQUENCED

Evento Trigger: Se decide cuando el avión podrá encender motores
Agente: ATC

Variables Iniciadas:

Variables Actualizadas:

Procesos Agentes:

Los agentes solo actuarán en caso de que haya alguna regulación o alguna actualización de tiempos (algún retraso), en caso de no haberlo los agentes solo actualizarán las variables que toque.
Para ver cómo los agentes actualizan los tiempos y propagan un retraso, recomendamos visitar el apartado 4.3 y el 5.5, donde se puede ver que agente actualiza cada tiempo y que relaciones hay entre los diferentes tiempos

CFMU	
ATC	Realiza predeparture sequence y a partir del TOBT del avión y el tráfico en el aeropuerto define el TSAT; actualiza status y tiempos si es necesario (Inicio Milestone)
AIRPORT	
Aircraft Operator	
Ground Handling	

Comentarios

- ATC introduce al avión en la pre-departure sequence y decide cuándo podrá iniciar taxi hacia pista. Si no hay problemas de congestión y se ha realizado una planificación correcta, no debería haber diferencia entre el EOBT y el TOBT.

A.2.11 Milestone 11

Definición: Embarque

Tiempo: ASBT (Actual Start Boarding Time)

Status avión: BOARDING

Evento Trigger: Empieza el embarque de pasajeros y carga en el avión

Agente: AO

Variables Iniciadas: ASBT pasa a ser “Actual”

Variables Actualizadas:

Procesos Agentes:

Los agentes solo actuarán en caso de que haya alguna regulación o alguna actualización de tiempos (algún retraso), en caso de no haberlo los agentes solo actualizarán las variables que toque.

Para ver cómo los agentes actualizan los tiempos y propagan un retraso, recomendamos visitar el apartado 4.3 y el 5.5, donde se puede ver que agente actualiza cada tiempo y que relaciones hay entre los diferentes tiempos

CFMU	
ATC	
AIRPORT	
Aircraft Operator	Inicia el embarque de pasajeros y carga en la aeronave. Actualiza status y tiempos si es necesario (Inicio Milestone).
Ground Handling	

Comentarios

- Este tiempo contempla si el avión empieza el embarque porque se tiene que ir nada más llegar al aeropuerto o si tiene que esperar unas horas por decisión de la aerolínea. En ambos casos el tiempo de embarque está relacionado con el tiempo en el que el avión quiere despegar, no con el tiempo en el que el avión ha llegado

A.2.12 Milestone 12

Definición: Aircraft Ready

Tiempo: ARDT (Aircraft Ready Time)

Status avión: READY

Evento Trigger: Cierre de puertas del avión

Agente: AO (Avión)

Variables Iniciadas: ARDT pasa a ser “Actual”

Variables Actualizadas:

Procesos Agentes:

Los agentes solo actuarán en caso de que haya alguna regulación o alguna actualización de tiempos (algún retraso), en caso de no haberlo los agentes solo actualizarán las variables que toque.

Para ver cómo los agentes actualizan los tiempos y propagan un retraso, recomendamos visitar el apartado 4.3 y el 5.5, donde se puede ver que agente actualiza cada tiempo y que relaciones hay entre los diferentes tiempos

CFMU	
ATC	
AIRPORT	
Aircraft Operator	El avión notifica que ya ha terminado el embarque y está listo para iniciar el vuelo. Actualiza status y tiempos si es necesario (Inicio Milestone)
Ground Handling	

Comentarios

- Este Milestone y el anterior suelen pasar seguidos, ya que en cuanto el avión cierra las puertas se lo considera "en vuelo" y los pilotos y la aerolínea quieren empezar las maniobras cuanto antes.
- Aircraft ready→ puertas cerradas, finger retirado y desconectado de los vehículos handling (listo para taxi inmediatamente)

A.2.13 Milestone 13

Definición: Start Up Approval Requested

Tiempo: ASRT (Actual Start Up Request Time)

Status avión: READY

Evento Trigger: Los pilotos piden autorización para encender motores

Agente: AO

Variables Iniciadas: ASRT pasa a ser "Actual"

Variables Actualizadas:

Procesos Agentes:

Los agentes solo actuarán en caso de que haya alguna regulación o alguna actualización de tiempos (algún retraso), en caso de no haberlo los agentes solo actualizarán las variables que toque.

Para ver cómo los agentes actualizan los tiempos y propagan un retraso, recomendamos visitar el apartado 4.3 y el 5.5, donde se puede ver que agente actualiza cada tiempo y que relaciones hay entre los diferentes tiempos

CFMU
ATC
AIRPORT

Aircraft Operator	El avión pide autorización para encender motores. Actualiza tiempos si es necesario (Inicio Milestone)
Ground Handling	

A.2.14 Milestone 14

Definición: Start Up Approved

Tiempo: ASAT

Status avión: READY

Evento Trigger: Se da autorización al avión para encender motores
Agente: ATC

Variables Iniciadas: TSAT pasa a ser ASAT (de Target a Actual)

Variables Actualizadas:

Procesos Agentes:

Los agentes solo actuarán en caso de que haya alguna regulación o alguna actualización de tiempos (algún retraso), en caso de no haberlo los agentes solo actualizarán las variables que toque.
Para ver cómo los agentes actualizan los tiempos y propagan un retraso, recomendamos visitar el apartado 4.3 y el 5.5, donde se puede ver que agente actualiza cada tiempo y que relaciones hay entre los diferentes tiempos

CFMU	
ATC	Se da autorización al avión a encender motores e iniciar las maniobras para iniciar el taxi. Actualiza tiempos si es necesario (Inicio Milestone).
AIRPORT	
Aircraft Operator	
Ground Handling	

Comentarios
- El avión es remolcado fuera de la plataforma para iniciar el taxi a pista.

A.2.15 Milestone 15

Definición: Off-Block (inicio taxi de salida)

Tiempo: AOBT

Status avión: OFF-BLOCK

Evento Trigger: Avión empieza el taxi a pista de despegue
Agente: ATC

Variables Iniciadas: EOBT pasa a ser AOBT (de Estimated a Actual)

Variables Actualizadas:

Procesos Agentes:

Los agentes solo actuarán en caso de que haya alguna regulación o alguna actualización de tiempos (algún retraso), en caso de no haberlo los agentes solo actualizarán las variables que toque.

Para ver cómo los agentes actualizan los tiempos y propagan un retraso, recomendamos visitar el apartado 4.3 y el 5.5, donde se puede ver que agente actualiza cada tiempo y que relaciones hay entre los diferentes tiempos

CFMU	
ATC	Detecta que el avión ha iniciado el taxi hacia la pista de despegue. Actualiza status y tiempos si es necesario (Inicio Milestone)
AIRPORT	
Aircraft Operator	
Ground Handling	

Comentarios

- Todo el proceso de garantizar que el avión llega a pista en el TTOT es responsabilidad del agente ATC, que con los datos de tiempos de taxi y tráfico que haber planeado correctamente la predeparture sequence

A.2.16 Milestone 16

Definición: Take-Off

Tiempo: ETOT

Status avión: DEPARTED

Evento Trigger: El avión despega
Agente: ATC

Variables Iniciadas: ETOT pasa a ser ATOT (de Estimated a Actual)

Variables Actualizadas:

Procesos Agentes:

Los agentes solo actuarán en caso de que haya alguna regulación o alguna actualización de tiempos (algún retraso), en caso de no haberlo los agentes solo actualizarán las variables que toque.

Para ver cómo los agentes actualizan los tiempos y propagan un retraso, recomendamos visitar el apartado 4.3 y el 5.5, donde se puede ver que agente actualiza cada tiempo y que relaciones hay entre los diferentes tiempos

CFMU	
ATC	Autoriza al avión a despegar del aeropuerto. Actualiza status y tiempos si es necesario (Inicio Milestone)
AIRPORT	
Aircraft Operator	
Ground Handling	

Comentarios
- Se autoriza al avión a despegar
- Finaliza el ACDM para este avión.

APÉNDICE B. DESARROLLO SISTEMA MULTIAGENTE

B.1 Introducción

Dada la complejidad que puede suponer el desarrollo un sistema multiagente se han ido creando y desarrollando diferentes metodologías con el fin de facilitar tanto una correcta creación del sistema cómo una fácil trazabilidad del mismo. De todas estas metodologías hemos optado por seguir la metodología GAIA, explicada con más detalle anteriormente.

Esta metodología, a partir de los requisitos establecidos por el cliente (en este caso nosotros mismos al definir el proyecto) nos permite obtener el correcto diseño de nuestro software.

A partir de seguir las fases de Análisis y Diseño, nos permite establecer el diseño del sistema multiagente. Obteniendo el número de agentes que se incluirán en la implementación final, las funciones que desempeñarán cada uno de ellos y las comunicaciones que se establecerán [11].

B.2 Fase Análisis

Primero deberemos realizar una fase de análisis con la intención de entender el sistema y su estructura.

Esta estructura viene dada por un conjunto de roles e interacciones (interactuación de los roles entre sí). Los roles a su vez se componen de responsabilidades (definen la funcionalidad), permisos o recursos (lectura, escritura y/o generación de información, elementos necesarios para llevar a cabo las responsabilidades) y protocolos (funciones que requieren de la interacción con otro rol) y actividades (no requieren de la interacción con otro roles).

Es necesario comentar que este documento plasma la versión final de los diferentes modelos. Durante el proceso proyecto ha sido necesario rehacer y modificar algunos roles, interacciones, agentes, etc. hasta llegar a la implementación final deseada. Durante la fase de desarrollo se ha tenido que volver en diversas ocasiones a la fase inicial de análisis.

B.2.1 Definición de roles

En este apartado se pretende definir los roles que encontramos en el sistema, para poder proceder definir los agentes del programa y las relaciones entre ellos.

Como ya hemos comentado, esta lista de roles es la versión final de los roles que encontramos en el programa, se ha tenido que ir actualizando a medida que se iba desarrollando el proyecto debido a que se iban actualizando los requisitos y la complejidad de los agentes.

La lista final de los roles en el programa es:

- Rol Broadcast información datos avión

- Rol actualización Tabla
- Rol enviar simulación a interfaz gráfica
- Rol enviar información output
- Rol decisión destinatario AO/GH
- Rol recibir/procesar parámetros
- Rol Trigger Milestones
- Rol controlar tiempo simulación
- Rol administrar lista Milestones + retrasos
- Rol finalizar simulación
- Rol administrar retrasos (tipo 1 y 4)
- Rol trigger retraso tipo 1 y 4
- Rol Cargar archivo Milestones
- Rol generar archivo Milestones resultados
- Rol cargar parámetros interfaz
- Rol enviar archivo tráfico aéreo a DDR2
- Rol cargar archivo input tráfico aéreo
- Rol cargar parámetros aeropuerto
- Rol definir parámetros iniciales
- Broadcast Parámetros Iniciales
- Rol generar archivos input
- Rol Broadcast archivos input
- Rol iniciar simulación
- Rol cargar retrasos simulación
- Rol definir retrasos simulación
- Rol Broadcast retrasos simulación
- Rol mostrar retrasos cargados + definidos
- Rol abrir interfaz resultados
- Rol generar archivo output parámetros
- Rol cargar archivo DDR2
- Rol generar archivo tráfico aéreo input
- Rol filtrar tráfico aéreo DDR2
- Rol definir tiempos de vuelo por *Waypoints*
- Rol relacionar tráfico aéreo de entrada y de salida
- Rol finalizar simulación gráficamente
- Rol mostrar simulación en tiempo real
- Rol mostrar resultados finales
- Rol Cargar resultados finales
- Rol cargar resultados comparación
- Rol mostrar comparación resultados
- Rol generar archivo output resultados
- Rol análisis resultados
- Rol mostrar análisis resultados
- Rol generar archivo output análisis resultados
- Rol cargar archivo tiempos turn around
- Rol guardar base de datos turn around
- Rol calculo tiempo turn around
- Rol envío tiempo turn around según contratos GH

- Rol publicar actualización datos avión
- Rol administrar tabla interna
- Rol definir/actualizar tiempos avión
- Rol administrar propagación retrasos avión
- Rol definir/actualizar Milestones avión
- Rol administrar lista retrasos
- Rol ejecutar retrasos tipo 2 y 3
- Rol ejecutar retrasos tipo 1 y 4
- Rol ejecutar retrasos estadísticos aleatorios
- Rol comunicación agente externo ACDM
- Rol comprobación ciclo turn around
- Rol cargar archivos input *Fligth Plans*
- Rol definir e introducir avión en el sistema CDM
- Rol iniciar ciclo CDM para un avión
- Rol definir categoría avión
- Rol cargar archivo vuelos por aerolínea
- Rol administrar pre-departure sequence
- Rol administrar retrasos tipo 4 (reducción capacidad de pista)

Es necesario comentar que algunos roles aparecerán en el programa más de una vez. Roles cómo los que cargan archivos, reciben parámetros o todos los roles relacionados con el envío y recepción de datos, estarán presentes en muchos de los agentes, por esta razón es posible que el modelo de interacciones no represente la totalidad del programa.

Para poder representar mejor las comunicaciones en el programa recomendamos ir a la sección de Diseño, al apartado del modelo de comunicaciones.

B.2.2 Modelo de interacciones

En este apartado GAIA define que hay que proceder a hacer un esquema gráfico de las comunicaciones que encontraremos entre los diferentes roles.

En este caso tenemos un gran número de roles y, cómo resultado, obtenemos un diagrama de interacciones muy grande y con un gran número de flechas.

Por esta razón hemos decidido no incluirlo en la memoria. No hemos visto necesario incluirlo ya que más adelante se hace una asignación de roles a agentes y un diagrama de comunicaciones entre estos, con este diagrama se podrá apreciar las comunicaciones entre los roles de una manera más sencilla y más intuitiva, puesto que una vez en el programa lo que nos interesa son las comunicaciones entre agentes, no entre roles.

B.2.3 Modelo de Roles

El modelo de roles nos permite establecer de manera detallada cuales son las funcionalidades (extraídas a partir de los requisitos) que debemos incluir en nuestro software.

Para explicar los roles es importante comentar que se componen de responsabilidades (definen la funcionalidad), permisos o recursos (lectura, escritura y/o generación de información, elementos necesarios para llevar a cabo las responsabilidades) y protocolos (funciones que requieren de la interacción con otro rol) y actividades (no requieren de la interacción con otro roles). En las responsabilidades definimos la de actuación (la función en sí, lo que hace el rol) y la seguridad (las comprobaciones que se llevan a cabo para garantizar que se ha realizado correctamente).

En este proyecto se ha considerado los mensajes cómo una parte importante de la simulación. Por esta razón se han incluido en la fase de análisis y se han tenido en cuenta a la hora de definir los roles. También se ha considerado la interacción del usuario con el programa cómo una parte importante a la hora de conocer diferentes datos e informaciones.

Es decir, en la siguiente explicación de los roles se ha considerado que en el apartado de “Recursos” de cada rol se tenga en cuenta su interacción con los mensajes y con las interfaces al usuario.
De esta manera se define:

- Lectura: Lectura de archivos (generalmente con la extensión *.txt*), contenido de mensajes recibidos y datos introducidos por el usuario a través de las interfaces gráficas.
- Escritura: Escritura de datos en archivos (tanto de resultados cómo inputs del programa generados por el mismo) y muestra de datos de la simulación o resultados al usuario a través de las interfaces gráficas.
- Generación: Generación de mensajes que serán enviados a otros agentes del programa.

A continuación encontramos la explicación de todos estos roles siguiendo esta estructura. Se han utilizado estas cajas para esquematizar la información y así facilitar la lectura y comprensión de los roles.

A continuación encontramos la explicación de todos estos roles siguiendo la estructura. Se han utilizado estas cajas para esquematizar los roles y así facilitar la lectura y comprensión de los roles.

Rol		**Broadcast información datos avión**
Descripción		Este rol es el encargado de preparar y enviar la información de una actualización de los datos de un avión del sistema.
Protocolos y actividades (Actuación)		Análisis actualización tabla, Codificación avión en mensaje, generación de lista de receptores, envío de mensaje.
Recursos	Lectura	-
	Escritura	-

	Generación	Generación y envío de mensajes para cada uno de los agentes que deben recibirlo.
	Seguridad	-

Rol		**Actualización Tabla**
Descripción		Recibe una nueva actualización y decodifica el mensaje. Comprueba la nueva información que llega al agente y actualiza únicamente los parámetros necesarios de cada avión.
Protocolos y actividades (Actuación)		Recepción mensaje, decodificación mensaje, análisis parámetros avión, análisis tabla, añadir avión a la tabla
Recursos	Lectura	Recepción, lectura y decodificación del mensaje con la información de los datos del avión actualizado.
	Escritura	-
	Generación	-
	Seguridad	-

Rol		**Enviar simulación a Interfaz Gráfica**
Descripción		Este rol es el encargado de preparar y enviar la información de una actualización de los datos de un avión a la interfaz Gráfica
Protocolos y actividades (Actuación)		Análisis actualización tabla, Codificación avión en mensaje, envío de mensaje.
Recursos	Lectura	-
	Escritura	-
	Generación	Generación y envío de mensajes al agente Simulador Gráfica.
	Seguridad	-

Rol		**Rol enviar información output**
Descripción		Este rol es el encargado de preparar y enviar la información de una actualización de los datos de un avión al Output. Para su guardado en el documento de resultados.
Protocolos y actividades (Actuación)		Análisis actualización tabla, Codificación avión en mensaje, envío de mensaje.
Recursos	Lectura	-
	Escritura	-
	Generación	Generación y envío de mensajes al agente Output.

Seguridad	-

Rol		**Rol decisión destinatario AO/GH**
Descripción		Este rol es el encargado de decidir que agente de Aerolínea o de GH será el encargado de recibir un mensaje. Se hace un análisis de la aerolínea a la que pertenece el avión para determinar AO; y una revisión de los contratos para decidir GH.
Protocolos y actividades (Actuación)		Análisis aerolínea del avión sobre el que se informa, análisis contratos de GH, análisis AO con agente propio, y selección de agentes que recibirán el mensaje.
Recursos	Lectura	-
	Escritura	-
	Generación	-
Seguridad		-

Rol		**Rol recibir/procesar parámetros**
Descripción		Recibe un mensaje con los parámetros de la simulación y decodifica el mensaje. Guarda la información del mensaje para poder aplicar los valores correspondientes durante la simulación. Cada agente seleccionará la información que le interesa para llevar a cabo la simulación
Protocolos y actividades (Actuación)		Recepción mensaje, decodificación mensaje, análisis parámetros simulación, guardado de parámetros.
Recursos	Lectura	Recepción, lectura y decodificación del mensaje con la información de los parámetros de simulación
	Escritura	-
	Generación	-
Seguridad		-

Rol	**Rol Trigger Milestones**
Descripción	Enviará un mensaje informando que un avión concreto ha llegado a un Milestone determinado. El destinatario depende del Milestone que se vaya a simular.
Protocolos y actividades (Actuación)	Análisis lista milestones + retrasos, análisis avión que realiza el milestone, búsqueda del

		receptor en función del Milestone y AO/GH del avión.
Recursos	Lectura	-
	Escritura	-
	Generación	Generación y envío del mensajes que avisa de que un avión ha realizado un milestone al agente correspondiente en cada caso.
	Seguridad	-

Rol		**Rol Controlar Tiempo Simulación**
Descripción		Este rol es el encargado de controlar el timer que decidirá en qué momento se envía un mensaje de que un avión ha realizado un nuevo Milestone. El parámetro que determinará la velocidad del timer viene definido por los parámetros iniciales.
Protocolos y actividades (Actuación)		Control timer para permitir o no avisar de la realización de un milestone por parte de un avión.
Recursos	Lectura	-
	Escritura	-
	Generación	-
	Seguridad	-

Rol		**Rol administrar lista milestones + retrasos**
Descripción		Recepción de las definiciones y actualizaciones de milestones y de la lista de retrasos. Administración de la lista, ordena cronológicamente para poder ejecutar la simulación correctamente. La lista de los retrasos se recibe una vez; pero las actualizaciones y definiciones de milestones se reciben constantemente, administración continuamente durante la simulación.
Protocolos y actividades (Actuación)		Recepción parámetros iniciales, recepción retrasos de la simulación, ordenación cronológica de los eventos. Recepción mensaje, decodificación mensaje, análisis hora del milestone, acoplamiento de la información a la lista de milestones + retrasos.
Recursos	Lectura	Recepción, lectura y decodificación de un mensaje con la información sobre la hora en la que sucede un Milestone

		Recepción y lectura de los mensajes con la lista de retrasos definidos para la simulación.
	Escritura	-
	Generación	-
	Seguridad	-

Rol		**Rol finalizar simulación**
Descripción		Este rol se encarga de mirar si se han realizado todos los milestones, en caso afirmativo mandar un mensaje de aviso al InfoSharing y al Output.
Protocolos y actividades (Actuación)		Análisis de los milestones no realizados. Cuando se terminan: mensaje de aviso de final de simulación a InfoSharing y Output
Recursos	Lectura	-
	Escritura	-
	Generación	Generación y envío de un mensaje que avisa de que se ha terminado la simulación.
	Seguridad	-

Rol		**Rol administrar retrasos tipo 1 y tipo 4**
Descripción		Este rol se encarga de leer y decodificar un mensaje que contiene información sobre la hora a la que sucederá un retraso. A continuación coloca el nuevo retraso en la posición correspondiente de la lista de milestones + retrasos (según orden cronológico)
Protocolos y actividades (Actuación)		Recepción mensaje, decodificación mensaje, análisis hora del retraso, acoplamiento de la información a la lista de milestones + retrasos.
Recursos	Lectura	Recepción, lectura y decodificación de un mensaje con la información sobre la hora en la que sucede un retraso
	Escritura	-
	Generación	-
	Seguridad	-

Rol	**Rol Trigger Retrasos tipo 1 y 4**
Descripción	Este rol es el encargado de preparar y enviar la información de que es la hora de realizar un retraso de tipo 1 o tipo 4

Protocolos y actividades (Actuación)		Análisis lista milestones + retrasos, análisis tipo de retraso, búsqueda de los receptores y envío de mensaje.
Recursos	Lectura	-
	Escritura	-
	Generación	Generación y envío del mensaje que avisa de que es la hora de aplicar un retraso de tipo 1 o tipo 4 a los agentes correspondientes.
Seguridad		-

Rol		**Rol Cargar archivo Milestones**
Descripción		Rol encargado de leer el documento TriggerCDM.txt con los primeros datos relacionados con los milestones que encontramos en los Flight Plans. También encontramos información relacionada con los retrasos tipo 1 y tipo 4.
Protocolos y actividades (Actuación)		Lectura documento TriggerCDM.txt, preparación de la lista con los milestones y los retrasos que se aplicarán durante la simulación.
Recursos	Lectura	Lectura de los tiempos de milestone y de los tiempos a los que se aplicarán los retrasos tipo 1 y tipo 4.
	Escritura	-
	Generación	
Seguridad		-

Rol		**Rol generar archivo Milestones Resultados**
Descripción		Rol encargado de escribir el documento MilestonesResultado.txt con la lista de milestones que ha ejecutado el programa.
Protocolos y actividades (Actuación)		Escritura del documento con la recopilación de los milestones simulados por el programa. El documento da información de milestones realizados y de horas de aplicación de retrasos.
Recursos	Lectura	-
	Escritura	-
	Generación	Generación documento con milestones simulados en el programa. Con la información de horas de aplicación de retrasos tipo 1 y tipo 4 y de hora de los milestones de cada avión.

Seguridad	-

Rol		**Rol cargar parámetros interfaz**
Descripción		Lectura y análisis de los parámetros iniciales que entran mediante la interfaz Simulator Input.
Protocolos y actividades (Actuación)		Lectura valores Interfaz Simulator Input, análisis valores, guardado de los parámetros.
Recursos	Lectura	Lectura de parámetros entrados por el usuario en la interfaz A-CDM Simulator Input.
	Escritura	-
	Generación	-
Seguridad		- Valores introducidos concuerdan con lo esperado (por ejemplo: Taxi Time = entero) - Comprobación que las rutas introducidas conducen a un archivo leíble. Si hay problemas con el formato o con encontrar el archivo el programa lo notifica.

Rol		**Rol enviar archivo tráfico aéreo DDR2.**
Descripción		Codificación mensaje con la ruta para encontrar el archivo con el tráfico aéreo que se simulará al agente DDR2. Envío de parámetros necesarios entrados por el usuario mediante el archivo Airport.txt
Protocolos y actividades (Actuación)		Codificación mensaje para el agente DDR2 con la ruta donde encontrar el documento con el tráfico aéreo y con parámetros cómo los puntos de la FIR, los puntos de APP y código ICAO a estudiar.
Recursos	Lectura	-
	Escritura	-
	Generación	Generación y envío del mensaje al agente DDR2 con toda la información necesaria para leer y entender correctamente el archivo con el tráfico aéreo que se estudiará.
Seguridad		Comprobación de que el archivo que se pide leer al agente DDR2 existe.

Rol	**Rol cargar archivo input tráfico aéreo.**
Descripción	Decodificación mensaje del DDR2 con aviso de que se ha creado el documento con la información sobre los aviones que se simularán y lectura del archivo con los datos.

Protocolos y actividades (Actuación)		Lectura y decodificación mensaje avisando de que el DDR2 ya ha terminado. Lectura del documento para un análisis de la información comprendida.
Recursos	Lectura	Recepción, lectura y decodificación de un mensaje con la información sobre si el DDR2 ha podido crear el documento input.txt Lectura del archivo con la información del tráfico aéreo que se simula.
	Escritura	-
	Generación	-
Seguridad		-

Rol		**Rol cargar parámetros aeropuerto**
Descripción		Lectura documento aeropuerto.txt para extraer parámetros que se utilizarán durante la simulación.
Protocolos y actividades (Actuación)		Lectura documento externo aeropuerto.txt con los parámetros que se utilizarán durante la simulación.
Recursos	Lectura	Lectura del archivo con la información de parámetros necesarios para la simulación
	Escritura	-
	Generación	-
Seguridad		Comprobación de que el documento existe con el nombre correcto.

Rol		**Rol definir parámetros iniciales**
Descripción		Acoplamiento de los parámetros iniciales definidos por interfaz y mediante el documento airport.txt. Los parámetros que se utilizaran en el programa.
Protocolos y actividades (Actuación)		Análisis parámetros iniciales definidos anteriormente por interfaz, análisis parámetros definidos mediante documento externo, unión de los parámetros en una sola lista de parámetros iniciales.
Recursos	Lectura	-
	Escritura	-
	Generación	-
Seguridad		-

Rol		**Rol Broadcast parámetros iniciales**
Descripción		Codificación mensaje con todos los parámetros necesarios para la simulación, contiene tanto los parámetros leídos en documento externo cómo los que se han introducido por interfaz. Envió del mensaje a todos los agentes que necesitarán utilizar alguno de estos parámetros.
Protocolos y actividades (Actuación)		Codificación mensaje con todos los parámetros, generación de lista de receptores, envío de mensaje.
Recursos	Lectura	-
	Escritura	-
	Generación	Generación y envío del mensaje que contiene todos los parámetros necesarios para la simulación.
	Seguridad	-

Rol		**Rol generar archivos input**
Descripción		Lectura del documento input.txt y creación de todos los documentos intermedios que utilizarán los diferentes agentes
Protocolos y actividades (Actuación)		Lectura documento input.txt, análisis datos, creación nuevos documentos input intermedios.
Recursos	Lectura	Lectura y análisis documento input.txt escrito por el agente DDR2.
	Escritura	Creación de todos los documentos input intermedios: - Flight Plans.txt - Flight Plans Salidas.txt - Trigger CDM.txt - AO1.txt - AO2.txt - Vuelos Aerolínea
	Generación	-
	Seguridad	-

Rol		**Rol Broadcast archivos input**
Descripción		Codificación mensaje con la ruta para encontrar los archivos input intermedios creados en el simulator input.
Protocolos y actividades (Actuación)		Codificación mensaje con la ruta donde encontrar los documentos input, generación de lista de receptores, envío de mensaje
Recursos	Lectura	-

	Escritura	-
	Generación	Generación y envío del mensaje a todos los agentes que necesitarán leer alguno de los documentos creados por el agente Simulator Input.
	Seguridad	-

Rol		**Rol iniciar Simulación**
Descripción		Codificación mensaje avisando de que se inicia la simulación.
Protocolos y actividades (Actuación)		Codificación mensaje, generación de lista de receptores, envío de mensaje
Recursos	Lectura	-
	Escritura	-
	Generación	Generación y envío del mensaje avisando de que se inicia la simulación.
	Seguridad	-

Rol		**Rol cargar retrasos simulación**
Descripción		Lectura de la ruta del documento de retrasos y lectura del documento con un posterior análisis de los datos que se encuentran en él.
Protocolos y actividades (Actuación)		Lectura desde la interfaz de la ruta documento externo con retrasos, lectura del documento, análisis de los datos.
Recursos	Lectura	Lectura de la interfaz para conseguir la ruta del archivo de retrasos. Lectura del documento con los retrasos que se aplicarán durante la simulación.
	Escritura	-
	Generación	-
	Seguridad	Comprobación de que el archivo que se pide leer existe, se encuentra en la ruta facilitada y tiene el formato correcto.

Rol	**Rol definir retrasos simulación**
Descripción	A través de la Interfaz Formulario de Retrasos se lleva a cabo la creación de un nuevo retraso. Se puede crear cualquier retraso en la simulación, añadiendo los parámetros del

		retraso a través de las diferentes opciones que brinda la interfaz.
Protocolos y actividades (Actuación)		Lectura desde la interfaz de los datos del nuevo retraso, análisis de estos datos, creación y definición del retraso en la lista de los retrasos que se aplicarán
Recursos	Lectura	Lectura de la interfaz para conseguir los datos de un nuevo retraso.
	Escritura	-
	Generación	-
Seguridad		- Comprobación de que los datos facilitados se pueden aplicar cómo un retraso a lo largo de una simulación. - Comprobación de que los datos introducidos concuerdan con el formato esperado. - Comprobación de que los tiempos de los diferentes retrasos tienen coherencia (Por ejemplo, Hora Conocimiento < Hora Inicio < Hora final)

Rol		**Rol Broadcast retrasos simulación**
Descripción		Codificación mensaje con la información de retrasos, análisis del retraso del cual se informa búsqueda del agente que aplicará el retraso, que será el agente receptor del mensaje. Se enviará un mensaje para cada uno de los retrasos.
Protocolos y actividades (Actuación)		Selección retraso a informar, análisis retraso, obtención del receptor, codificación mensaje y envío de este.
Recursos	Lectura	-
	Escritura	-
	Generación	Generación y envío del mensaje con la información relacionada a un retraso que se aplicará durante la simulación
Seguridad		-

Rol	**Rol mostrar retrasos simulación, cargados + definidos**
Descripción	Análisis de la totalidad de los retrasos que se aplicaran, tanto los definidos por interfaz cómo los leídos de un documento externo y clasificación de estos retrasos por interfaz. Dando la posibilidad de modificar la lista de

		retrasos añadiendo o borrando cualquiera de ellos.
Protocolos y actividades (Actuación)		Análisis retrasos, escritura de los retrasos en interfaz, separándolos por tablas según el tipo de retraso, borrado de retrasos y administración de la lista de retrasos
Recursos	Lectura	-
	Escritura	Escritura en las tablas de retrasos definidos en interfaces y/o leídos en un documento externo.
	Generación	-
Seguridad		-

Rol		**Rol abrir interfaz resultados**
Descripción		Avisa al agente Output y la interfaz Output de que se procede a abrir la interfaz y cargar unos resultados. Se añade la ubicación del documento de resultados
Protocolos y actividades (Actuación)		Codificación y envío del mensaje para la apertura de la interfaz de resultados.
Recursos	Lectura	-
	Escritura	-
	Generación	- Generación del mensaje que avisa que hay que abrir la interfaz resultados. Contiene la información de la ubicación del documento de resultados que se mostrará
Seguridad		- Comprobación de que en la ubicación establecida se encuentra el documento de resultados

Rol		**Rol generar archivo output parámetros**
Descripción		Análisis de los parámetros definidos en el programa. Escritura de estos parámetros en un archivo externo.
Protocolos y actividades (Actuación)		Recopilación parámetros que se utilizan y escritura de estos en un documento externo
Recursos	Lectura	-
	Escritura	Creación y escritura de un documento externo que contendrá una lista con los parámetros que se utilizarán durante la simulación.
	Generación	-
Seguridad		--

Rol		**Rol cargar archivo DDR2**
Descripción		Decodificación mensaje con la ruta para encontrar el archivo con el tráfico aéreo que se simulará y lectura del archivo.
Protocolos y actividades (Actuación)		Lectura y decodificación mensaje con la ruta del documento con la información DDR2. Lectura del documento para un análisis de la información comprendida.
Recursos	Lectura	Recepción, lectura y decodificación de un mensaje con la información sobre la ruta de un archivo. Lectura del archivo con la información del tráfico aéreo que se simula.
	Escritura	-
	Generación	-
Seguridad		-

Rol		**Rol filtrar tráfico aéreo DDR2**
Descripción		Análisis de la información adquirida al leer el documento DDR2 y filtrado según si pertenecen al aeropuerto de estudio.
Protocolos y actividades (Actuación)		Análisis datos, definición de los vuelos que encontramos en el documento DDR2 y que pertenecen al aeropuerto de estudio.
Recursos	Lectura	-
	Escritura	-
	Generación	-
Seguridad		-

Rol		**Rol definir tiempos de vuelo por Waypoints**
Descripción		Análisis de la información adquirida al definir los vuelos de nuestro aeropuerto y definición de los tiempos TFIR y TAPP de cada vuelo en función de los waypoints por los que vuela el avión
Protocolos y actividades (Actuación)		Análisis datos, definición de los TFIR i TAPP de cada vuelo estudiado.
Recursos	Lectura	-
	Escritura	-
	Generación	-
Seguridad		-

Rol		**Rol relacionar tráfico aéreo de entrada y de salida**
Descripción		Análisis de los vuelos de estudio para definir las posibles relaciones entre llegadas y salidas. El objetivo es encontrar en qué casos un vuelo de una llegada y una salida podrían ser realizados por el mismo avión.
Protocolos y actividades (Actuación)		Análisis datos, búsqueda de relaciones vuelos llegadas y salidas en función de los parámetros de cada vuelo.
Recursos	Lectura	-
	Escritura	-
	Generación	-
Seguridad		-

Rol		**Rol generar archivo tráfico aéreo input**
Descripción		Escritura documento intermedio con la información de todos los aviones que se simularán según la información analizada anteriormente proveniente del documento DDR2.
Protocolos y actividades (Actuación)		Escritura del documento con la información de los vuelos que se simularán.
Recursos	Lectura	Creación y escritura documento input.txt
	Escritura	-
	Generación	-
Seguridad		-

Rol		**Rol finalizar simulación gráficamente**
Descripción		Activación al pulsar el botón de finalizar simulación. Codificación del mensaje avisando de que se termina la simulación y envío del mensaje.
Protocolos y actividades (Actuación)		Escucha de botón, codificación mensaje avisando fin de simulación y envío de mensaje.
Recursos	Lectura	Escucha de botón.
	Escritura	-
	Generación	Generación y envío del mensaje avisando de que se ha terminado la simulación prematuramente.
Seguridad		-

Rol		**Rol mostrar simulación en tiempo real**
Descripción		Lectura y decodificación de mensajes avisando de alguna actualización en algún avión y escritura de la nueva información en la tabla mostrada por interfaz.
Protocolos y actividades (Actuación)		Lectura y decodificación mensaje, análisis información, análisis tabla mostrada, escritura de la nueva información por pantalla
Recursos	Lectura	Lectura y decodificación mensaje avisando de una actualización en algún avión.
	Escritura	Escritura de los nuevos datos por interfaz a tiempo real mediante una tabla.
	Generación	-
Seguridad		-

Rol		**Rol mostrar resultados finales**
Descripción		Análisis tabla con los resultados de una simulación y escritura de los datos por pantalla.
Protocolos y actividades (Actuación)		Análisis datos y escritura mediante interfaz de los datos encontrados en la tabla de aviones.
Recursos	Lectura	-
	Escritura	Escritura por pantalla de todos los datos sobre los aviones de una simulación.
	Generación	-
Seguridad		-

Rol		**Rol cargar resultados finales**
Descripción		Carga de los resultados finales de la simulación desde el archivo en el que se han guardado
Protocolos y actividades (Actuación)		Lectura del archivo de resultados, decodificación, creación de tabla interna y muestra de la tabla por interfaz
Recursos	Lectura	Lectura del archivo de resultados
	Escritura	Escritura por pantalla de todos los datos sobre los aviones de una simulación.
	Generación	-
Seguridad		-

Rol		**Rol cargar resultados finales comparación**
Descripción		Carga de resultados de otra simulación para su posterior comparación, a través de cargar el archivo de resultados que generó el programa en otra simulación.
Protocolos y actividades (Actuación)		Lectura desde la interfaz de la ruta, análisis de esta, apertura del documento y extracción y análisis de la información encontrada.
Recursos	Lectura	Lectura de la interfaz para conseguir la ruta donde encontrar el documento de retrasos Lectura del documento resultados finales comparación.
	Escritura	-
	Generación	-
Seguridad		- Comprobación de que el documento existe en la ruta indicada - Comprobación de formato del archivo (no sería necesaria porque se trata directamente del archivo de resultados de otra simulación)

Rol		**Rol mostrar comparación resultados**
Descripción		Muestra por pantalla de dos tablas, la simulación acabada y los resultados cargados. Tablas en el mismo formato para permitir la comparación de datos por parte del usuario.
Protocolos y actividades (Actuación)		Análisis datos y escritura mediante interfaz de los datos encontrados en cada una de las tablas de tráfico aéreo.
Recursos	Lectura	-
	Escritura	Escritura de dos tablas con los datos del tráfico aéreo simulado en ambos resultados
	Generación	-
Seguridad		-

Rol		**Rol generar archivo output resultados**
Descripción		Creación y escritura del documento externo con toda la información de los aviones al final de la simulación al recibir el mensaje de que se ha terminado la simulación. Cada línea un avión. Codificado de la misma manera que los mensajes.
Protocolos y actividades (Actuación)		Análisis tabla, codificación tráfico aéreo, escritura del documento de salida con los resultados de la simulación

Recursos	Lectura	-
	Escritura	Escritura del documento externo donde cada línea del documento contendrá la información de un avión simulado
	Generación	-
Seguridad		-

Rol		**Rol análisis resultados**
Descripción		Análisis de la tabla resultados y análisis del documento externo leído con resultados si se da el caso. Da la posibilidad al usuario de elegir que parámetro de los aviones estudiar, así cómo utilizar una ventana de tiempo de estudio y un filtro para seleccionar que tipos de aviones le interesan.
Protocolos y actividades (Actuación)		Filtrado de aviones por AO y modelo y utilización ventana de tiempo, selección de parámetro, análisis aviones en la tabla, generación de las variables y los registros debido al análisis.
Recursos	Lectura	-
	Escritura	-
	Generación	-
Seguridad		-

Rol		**Rol mostrar análisis resultados**
Descripción		Escritura de los datos analizados por pantalla. Se muestran variables cómo el número de aviones analizados, el número de minutos totales de delay, un registro de aviones en delay por minuto, etc.
Protocolos y actividades (Actuación)		Escritura mediante interfaz gráfica de los datos analizados sobre los resultados.
Recursos	Lectura	-
	Escritura	Escritura por pantalla de los datos resultantes de los análisis de los resultados de las simulaciones.
	Generación	-
Seguridad		-

Rol		**Rol generar archivo output análisis resultados**
Descripción		Creación y escritura del documento externo con toda la información recopilada al hacer el análisis de los resultados
Protocolos y actividades (Actuación)		Análisis variables resultantes, codificación variables y escritura del documento de salida con los resultados encontrados al realizar el análisis de los resultados
Recursos	Lectura	-
	Escritura	Escritura del documento externo con los resultados definidos al realizar el análisis de los resultados
	Generación	-
Seguridad		-

Rol		**Rol cargar archivos tiempos turn around**
Descripción		Lectura del documento turn around, y análisis de los datos
Protocolos y actividades (Actuación)		Apertura del documento y análisis de la información encontrada.
Recursos	Lectura	Lectura del documento con los tiempos de Turn Around de los aviones.
	Escritura	-
	Generación	-
Seguridad		Comprobación de que el documento existe en la ruta indicada.

Rol		**Rol guardar base de datos turn-around**
Descripción		Guardado en la base de datos de los resultados leídos sobre los tiempos de turn around de los diferentes aviones.
Protocolos y actividades (Actuación)		Guardado de los datos en la base de datos del agente.
Recursos	Lectura	-
	Escritura	-
	Generación	-
Seguridad		-

Rol		**Rol calculo tiempo turn around**
Descripción		Calculo del tiempo de turna round del avión pedido. Recepción de mensaje con la solicitud.
Protocolos y actividades (Actuación)		Lectura y decodificación mensaje con el avión a estudiar, análisis de la base de datos, extracción tiempo de turn around
Recursos	Lectura	Recepción, lectura y decodificación de un mensaje con la información sobre el avión a estudiar.
	Escritura	-
	Generación	-
	Seguridad	-

Rol		**Rol envío tiempo turn around según contratos GH**
Descripción		Este rol es el encargado de preparar y enviar la información con el tiempo de turn around de un avión en concreto al agente GH que tiene el contrato para la aerolínea del avión estudiado.
Protocolos y actividades (Actuación)		Análisis contratos, decisión del agente receptor del mensaje, codificación del mensaje y envío de este.
Recursos	Lectura	-
	Escritura	-
	Generación	Generación y envío de mensajes para el agente con el contrato del avión estudiado.
	Seguridad	-

Rol		**Rol publicar actualización datos avión**
Descripción		Envío de los datos de un avión al InfoSharing cuando ha habido una actualización. También se añade la opción de mostrar por consola los datos enviados.
Protocolos y actividades (Actuación)		Análisis de los datos del avión, codificación y envío del mensaje al InfoSharing y escritura por consola de los datos (si se quiere)
Recursos	Lectura	-
	Escritura	Escritura por consola de los datos del avión actualizado
	Generación	Envío del mensaje con la actualización de los datos del avión al InfoSharing

Seguridad	-

Rol		**Rol administrar tabla interna**
Descripción		Este rol se encarga de decidir en qué puesto de la tabla se guardara cada avión. Los nuevos aviones siempre ocuparán la primera posición vacía que se encuentre en la tabla. Pero este rol analizará en cada caso si un avión ya existía con el objetivo de asegurar que no tenemos 2 veces el mismo avión en la tabla.
Protocolos y actividades (Actuación)		Análisis avión que se quiere insertar, análisis tabla existente, decisión de donde colocar el avión y guardado de la información.
Recursos	Lectura	-
	Escritura	-
	Generación	-
Seguridad		-

Rol		**Rol definir/actualizar tiempos avión**
Descripción		Analiza los datos recibidos de un avión y decide si es necesario actuar y calcular alguno de los demás datos. (Por ejemplo: si se ha actualizado el ELDT se tiene que calcular un nuevo EIBT)
Protocolos y actividades (Actuación)		Decodificación mensaje actualización avión, análisis parámetros y actualización avión, cálculo de los nuevos tiempos.
Recursos	Lectura	Recepción y decodificación del mensaje con la actualización de los datos de un avión. (Normalmente Broadcast Actualización Avión)
	Escritura	-
	Generación	-
Seguridad		-

Rol	**Rol administrar propagación avión**
Descripción	Define la propagación de los retrasos. Tiene definidas relaciones entre los diferentes tiempos de un avión para propagar un retraso de manera realista. (Por ejemplo: si se retrasa el tiempo de inicio de taxi se retrasará el tiempo de despegue)

Protocolos y actividades (Actuación)		Análisis datos de un avión, análisis relaciones entre tiempos, actualización y cálculo de los nuevos tiempos.
Recursos	Lectura	-
	Escritura	-
	Generación	-
Seguridad		-

Rol		**Rol definir/actualizar milestones avión**
Descripción		Este rol es el encargado de definir o actualizar el tiempo de un milestone de un avión en función de la información que tiene de este. Esta Milestone definido/actualizado se envía al MilestoneTrigger para que modifique su lista de milestones en la simulación.
Protocolos y actividades (Actuación)		Análisis datos avión, análisis relaciones tiempos-milestones, actualización milestones avión, codificación y envío de mensaje a MT.
Recursos	Lectura	-
	Escritura	-
	Generación	Generación de mensaje con la actualización de un milestone de un avión concreto.
Seguridad		-

Rol		**Rol administrar lista retrasos**
Descripción		Este rol es el encargado de añadir y organizar la lista con todos los retrasos (tipo 1, 2 y 3) que recibirán los aviones durante la simulación.
Protocolos y actividades (Actuación)		Recepción y decodificación de mensaje con retraso, análisis del nuevo retraso, comprobación de que es correcto, añadir retraso a la lista del agente.
Recursos	Lectura	Recepción y decodificación de mensaje con el retraso
	Escritura	-
	Generación	-
Seguridad		Se comprueba que el retraso que se añadirá sea posible y que no fueran definidos anteriormente.

Rol		**Rol ejecutar retrasos tipo 2 y tipo 3**
Descripción		Ejecuta un retraso tipo 2 o 3 de un avión concreto al detectar que se ha llegado al momento adecuado. Esta detección se realiza al recibir un mensaje de actualización de datos referido a actualización de milestone del avión.
Protocolos y actividades (Actuación)		Análisis datos, actualización avión análisis base interna de retrasos por aplicar. Aplicación retraso cuando toque, análisis del retraso, codificación mensaje selección receptor y envío mensaje.
Recursos	Lectura	-
	Escritura	-
	Generación	Generación y codificación de un mensaje con los nuevos parámetros de un avión
	Seguridad	-

Rol		**Rol ejecutar retrasos tipo 1 y tipo 4**
Descripción		Ejecución de un retraso tipo 1 o 4. Estos se ejecutan al llegar un momento concreto de la simulación, cuando MilestoneTrigger detecta que se ha llegado al momento del retraso y avisa al agente que toca. Este rol se encarga de la decodificación de un mensaje con la información de una hora análisis de esta información y análisis de la lista interna con los retrasos que se tendrán de aplicar durante la simulación. A continuación se decide si se aplica algún retraso, en caso afirmativo se modifican los valores necesarios y se avisa al InfoSharing de cada uno de los aviones que se hayan modificado.
Protocolos y actividades (Actuación)		Decodificación mensaje, análisis datos, análisis base interna de retrasos por aplicar. Aplicación retraso cuando toque a todos los aviones que resulten afectados, codificación mensajes selección receptor y envío mensaje.
Recursos	Lectura	Recepción, lectura y decodificación de un mensaje con la información sobre un avión.
	Escritura	-
	Generación	Generación y codificación de mensajes con los nuevos parámetros de un avión. Si afecta a más de un avión = un mensaje por cada avión afectado.
	Seguridad	-

Rol		**Rol ejecutar retrasos estadísticos aleatorios**
Descripción		Este rol es el encargado de ejecutar un retraso estadístico y aplicar el retraso decidido en cada caso. (Puede estar activado o desactivado)
Protocolos y actividades (Actuación)		Análisis de un avión y de los parámetros de simulación para saber si esta activada la opción. En caso afirmativo: análisis parámetros y decisión del retraso que se aplica. Ejecución del retraso seleccionado en cada caso. Si se ha aplicado retraso, generación codificación y decisión del receptor (InfoSharing) de un mensaje avisando de la actualización.
Recursos	Lectura	-
	Escritura	-
	Generación	Generación y codificación de mensajes con los nuevos parámetros de un avión cada uno
	Seguridad	El programa no permite adelantar un tiempo en ningún caso. Solo retrasos.

Rol		**Rol comunicación agente externo ACDM**
Descripción		Este rol es el encargado de preparar y enviar un mensaje solicitando el tiempo de turn around de un avión en concreto.
Protocolos y actividades (Actuación)		Análisis avión del que se requiere información, codificación mensajes selección receptor y envío mensaje.
Recursos	Lectura	-
	Escritura	-
	Generación	Generación y envío de un mensaje al agente Turn Around.
	Seguridad	-

Rol		**Rol comprobación ciclo turn around**
Descripción		Comprueba se cumplen los tiempos turn around. Comprueba que entre llegada y salida de plataforma hay un tiempo mínimo de turn around.
Protocolos y actividades (Actuación)		Análisis actualización tiempos, análisis de tiempo turn around recibido, actualización de tiempos, aviso de actualización de tiempos.
Recursos	Lectura	-

	Escritura	-
	Generación	-
	Seguridad	-

Rol		**Rol cargar archivos input Flight Plans**
Descripción		Lectura del documento de Flight Plans, y análisis de los datos
Protocolos y actividades (Actuación)		Apertura del documento y análisis de la información encontrada.
Recursos	Lectura	Lectura del documento con los valores que encontramos en un plan de vuelo de un avión
	Escritura	-
	Generación	-
	Seguridad	-

Rol		**Rol definir e introducir avión en el Sistema CDM**
Descripción		Análisis de los datos y cómo ese avión no había aparecido antes análisis de la información leída en los planes de vuelo para crear este avión con los parámetros pertenecientes.
Protocolos y actividades (Actuación)		Análisis datos mensaje, análisis datos leídos en documento externo (planes de vuelo), definición del avión siguiendo una clase Aircraft.
Recursos	Lectura	-
	Escritura	-
	Generación	-
	Seguridad	-

Rol		**Rol Iniciar ciclo CDM para un avión**
Descripción		Cuando aparece un nuevo avión en el sistema avisamos al InfoSharing para que realice un Broadcast avisando a todos. Este avión es la primera vez que ha aparecido en el simulador.
Protocolos y actividades (Actuación)		Generación codificación y decisión del receptor (InfoSharing) de un mensaje avisando de la aparición de un nuevo avión.
Recursos	Lectura	-

	Escritura	-
	Generación	Generación y envío de un mensaje al agente InfoSharing informando del nuevo avión
	Seguridad	-

Rol		**Rol definir categoría avión**
Descripción		Análisis avión para obtener la categoría de cada uno en función de los parámetros de simulación donde venía la base de datos que se utiliza con una lista de modelos y categorías.
Protocolos y actividades (Actuación)		Análisis avión y definición del parámetro categoría en función de la base de datos.
Recursos	Lectura	-
	Escritura	-
	Generación	-
	Seguridad	-

Rol		**Rol cargar archivos vuelos por Aerolínea**
Descripción		Lectura del documento de vuelos Aerolínea, y análisis de los datos
Protocolos y actividades (Actuación)		Apertura del documento y análisis de la información encontrada.
Recursos	Lectura	Lectura del documento con la información sobre todas las llegadas y salidas del aeropuerto.
	Escritura	-
	Generación	-
	Seguridad	-

Rol		**Rol administrar pre-departure sequence**
Descripción		Administra el orden en el que los aviones aterrizarán o despegarán. Con los parámetros de operaciones/hora para aterrizajes y despegues se ordenan los aviones de manera que adecuar la frecuencia de operaciones.
Protocolos y actividades (Actuación)		Análisis parámetros escogidos para pre-departure sequence. Análisis de la secuencia definida hasta el momento, definición nuevos horarios de llegadas y salidas, si se ha modificado algún tiempo aviso al InfoSharing,

		generación codificación y decisión del receptor (InfoSharing) de un mensaje avisando de la actualización.
Recursos	Lectura	-
	Escritura	-
	Generación	Generación y envío de un mensaje al agente InfoSharing informando de una actualización
Seguridad		-

Rol		**Rol administrar retrasos tipo 4 (reducción capacidad de pista)**
Descripción		Se lleva a cabo un retraso que consiste en reducir el parámetro de operaciones/hora y llevar a cabo una nueva pre-departure sequence.
Protocolos y actividades (Actuación)		Análisis del nuevo retraso, comprobación de que es correcto, y finalmente se añade a la lista de los retrasos que afectarán a la capacidad de pista.
Recursos	Lectura	-
	Escritura	-
	Generación	-
Seguridad		Se comprueba que el retraso que se añadirá sea posible y que no sea repetido

En este conjunto de roles encontramos algunos que se han simplificado para no aumentar demasiado el número de roles.
Los roles de actualizar información y Milestones deberían de encontrarse más veces, puesto que el proceso a seguir para definir/actualizar cada uno de los Milestones es diferente, pero debido a que esta diferencia no supone un cambio suficientemente grande cómo para que sea necesario añadir 16 roles a la lista que ya hay de 65, hemos optado por juntar todos estos roles en uno y explicar la diferencia en el apartado de explicación de los agentes, igual que en los roles de actualizar y calcular los diferentes datos de un avión.

B.3 Fase de Diseño

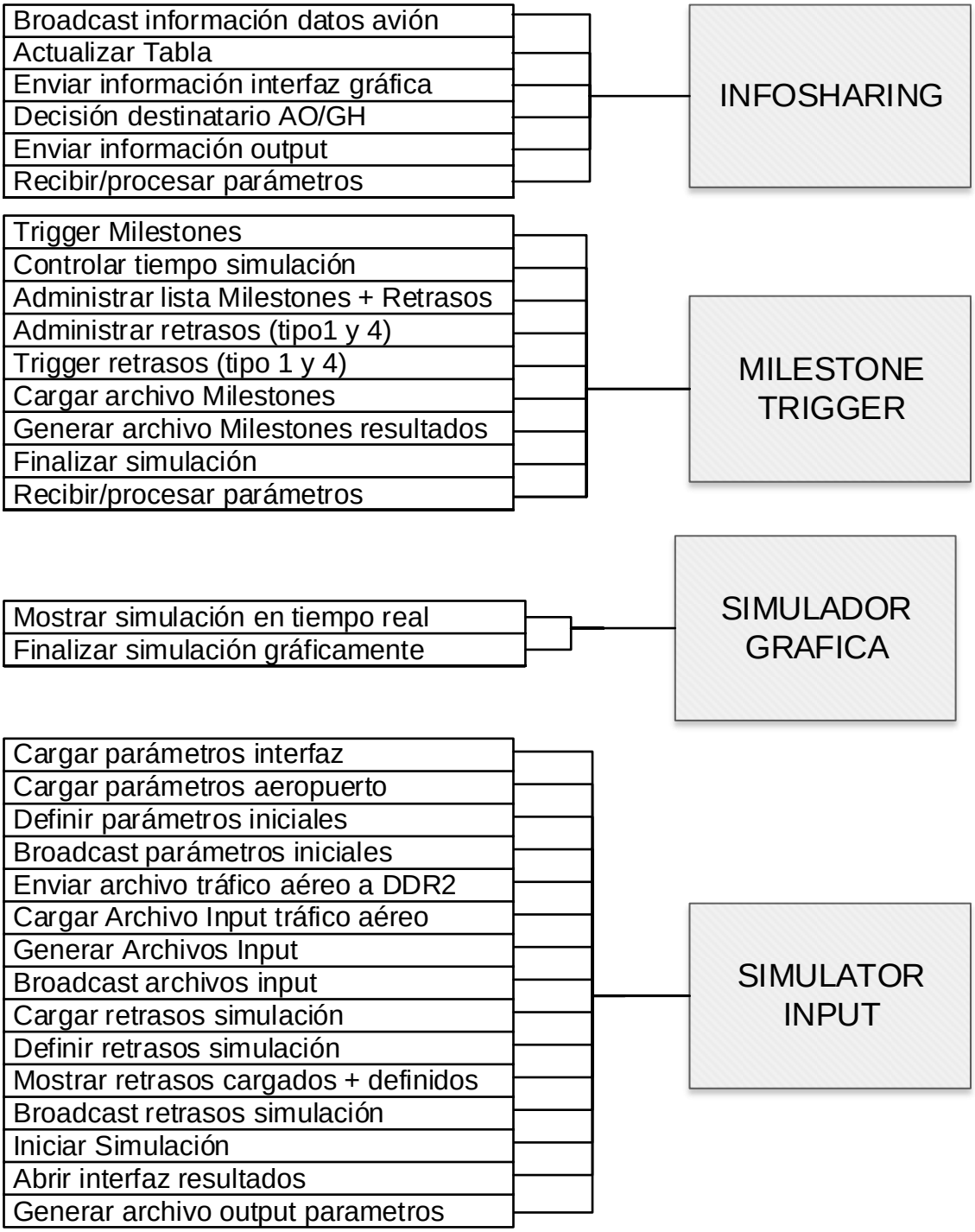
B.3.1 Modelo de Agentes

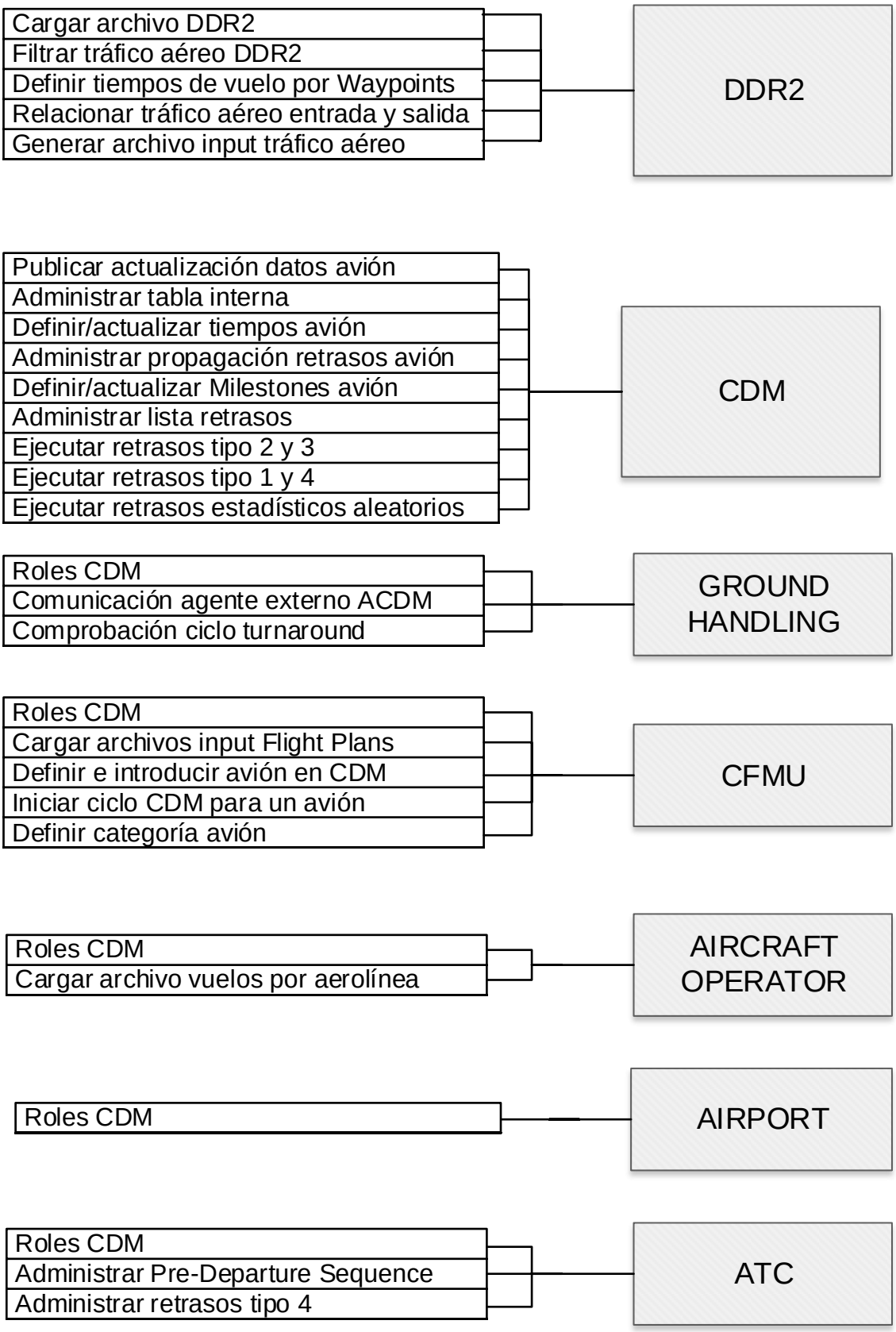
En este modelo se muestran los agentes definidos a partir del análisis de roles hecho previamente. En este diagrama se aprecian tanto los agentes definidos por el manual de Eurocontrol [4], los conocidos cómo “Agentes A-CDM”, cómo los creados para que el programa funcione y cumpla con los requisitos del proyecto.

Para este proyecto en particular, el manual de implementación A-CDM nos define una serie de agentes que obligatoriamente se tienen que poner en el programa.

Este caso hace que no se pueda seguir la metodología GAIA al pie de la letra *[Roles → Agrupación de roles → Agentes]*, sino que para los agentes definidos cómo "Agentes A-CDM" se seguirá *[Agentes ACDM → Asignación de roles]* y para los demás se ha seguido la metodología habitual.

Gracias a este modelo podemos ver qué rol desempeña cada agente, que se han formado a partir de la agrupación de roles definidos en el apartado de análisis. Y podemos apreciar en la figura B-1 cómo finalmente se ha optado por implementar 13 agentes.




Cargar archivo DDR2
Filtrar tráfico aéreo DDR2
Definir tiempos de vuelo por Waypoints
Relacionar tráfico aéreo entrada y salida
Generar archivo input tráfico aéreo
DDR2
Publicar actualización datos avión
Administrar tabla interna
Definir/actualizar tiempos avión
Administrar propagación retrasos avión
Definir/actualizar Milestones avión
Administrar lista retrasos
Ejecutar retrasos tipo 2 y 3
Ejecutar retrasos tipo 1 y 4
Ejecutar retrasos estadísticos aleatorios
CDM
Roles CDM
Comunicación agente externo ACDM
Comprobación ciclo turnaround
GROUND HANDLING
Roles CDM
Cargar archivos input Flight Plans
Definir e introducir avión en CDM
Iniciar ciclo CDM para un avión
Definir categoría avión
CFMU
Roles CDM
Cargar archivo vuelos por aerolínea
AIRCRAFT OPERATOR
Roles CDM
AIRPORT
Roles CDM
Administrar Pre-Departure Sequence
Administrar retrasos tipo 4
ATC

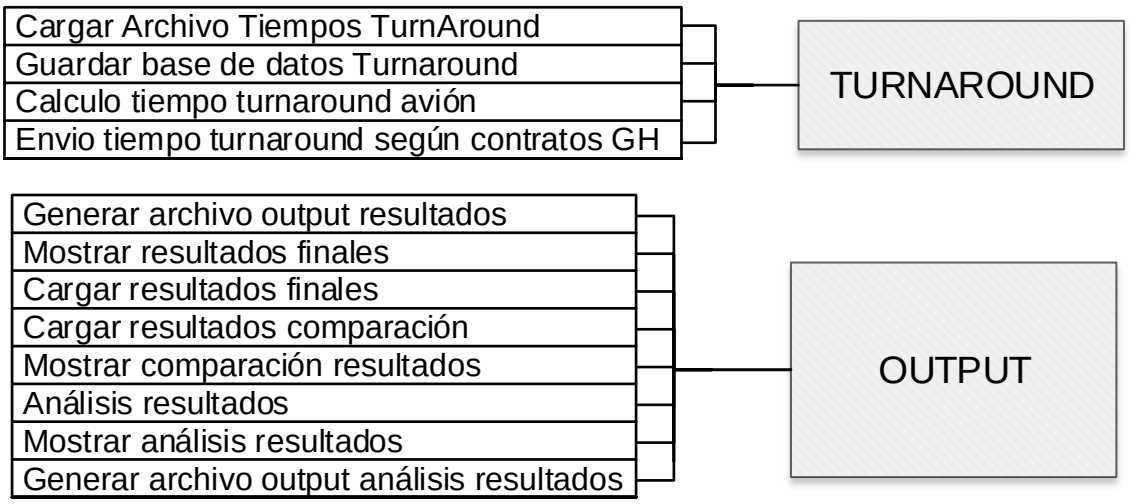


Figura B-1: Diagramas de Agentes

Tal y cómo se ha especificado anteriormente, los “Agentes ACDM” tienen una estructura igual, y se diferencian en a que *stakeholder* del aeropuerto simulan y, por tanto, en los roles que desempeñan en el sistema A-CDM. Este hecho hace que compartan una serie de roles e interacciones primarias, cómo “Actualizar tabla” o “Publicar Información”. Por esta razón se ha evitado repetir 5 veces cada uno de estos Roles e Interacciones y se han colocado al agente “CDM”. Este agente es un agente ficticio que se utilizará para explicar los roles, interacciones y propiedades que comparten los 5 “Agentes ACDM”.

Estos 5 “Agentes ACDM” (CFMU, Airport, ATC, AO y GH) se diferencian básicamente en que tiempos del avión actualizan, que Milestones se encargan de simular y en algunas funcionalidades extra que ofrece el programa. Por tanto comparten los conocidos cómo “roles CDM”, aunque cada de ellos aplique estos roles de manera diferente.

Estas diferencias en la aplicación de estos roles por parte de estos 5 agentes están explicadas en el apartado de explicación de agentes dentro de la memoria.

También es necesario comentar que debido a que los agentes AO y GH representan una sola aerolínea y empresa de handling respectivamente, en la ejecución del programa tendremos que definirlos múltiples veces, una por cada aerolínea o empresa de handling.

Este hecho hace que al definir los agentes cuando se va a simular, podamos tener múltiples AO y GH, y el número de agentes en el programa aumente considerablemente. Por ejemplo, en las últimas simulaciones hechas, para comprobar que el programa funcionase, se utilizaron 12 aerolíneas (12 agentes AO) y 5 empresas de handling (5 agentes GH); por tanto la simulación fue de 27 agentes.

A partir de este modelo de roles se puede deducir la función de cada agente dentro del programa. Estas funcionalidades no se explicarán en este apartado, ya que programar estos agentes es un paso posterior a este análisis; una

explicación más detallada de estos agentes se puede encontrar en el apartado correspondiente en esta misma memoria.

B.3.2 Modelo de Comunicaciones

Este modelo, figura B-2, muestra cuáles serán las relaciones entre los agentes durante la ejecución del programa, es decir, nos enseña entre que agentes habrá mensajes.
Este modelo es fácil de definir gracias a que previamente se ha hecho una fase de análisis de las interacciones entre roles y una asignación de roles a agentes.

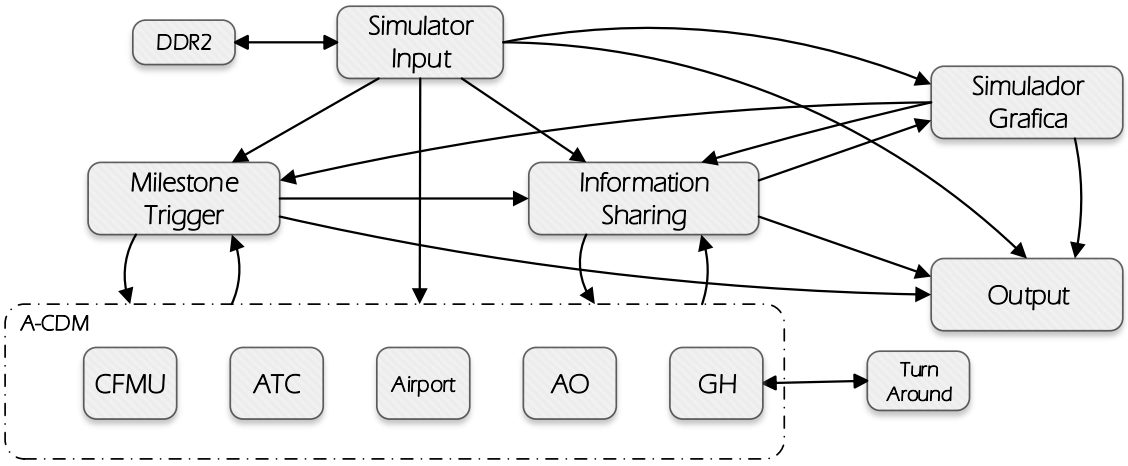


Figura B-2: Diagrama de Comunicaciones

En el diagrama de la figura B-2 se aprecia cómo “desaparece” el agente CDM, que sí aparecía en los modelos anteriores. Esto es debido a que al no ser un agente al uso, sino un agente del que heredan los “Agentes ACDM”, no tiene una comunicación establecida con ninguno de los otros agentes del programa, no se ejecuta en el programa cómo tal.

El hecho de que algunos agentes se comuniquen con todos los agentes que heredan de CDM, y que con cuál de ellos se comunique cada vez dependa del contenido del mensaje, se ha representado en este diagrama cómo una comunicación al “cuadrado” que engloba a los 5 “Agentes ACDM”. Las comunicaciones concretas entre un agente y uno de los “Agentes ACDM” se han representado con normalidad, con un conector directo entre los agentes.
Este uso de la agrupación de los agentes CDM ha hecho que se vea un diagrama mucho más limpio y con menos flechas, pues cada una de las flechas a o desde CDM debería de estar 5 veces.

Modelo de comunicaciones con interfaces

La metodología GAIA solo nos pide la comunicación entre agentes al uso. Pero tal y cómo se ha planteado este proyecto, el programa contará con una serie de interfaces.
Estas interfaces están pensadas para facilitar al usuario el contacto directo con diferentes aspectos de la simulación, cómo ver la simulación en directo, definir diversos parámetros de la simulación, administrar los retrasos, etc.

Estas comunicaciones entre agentes e interfaces son importantes para comprender el funcionamiento del programa, y están directamente relacionadas con algunos roles de los agentes.

Para la figura B-3 se ha llevado a cabo un nuevo análisis de comunicaciones, añadiendo las comunicaciones entre agentes e interfaces.

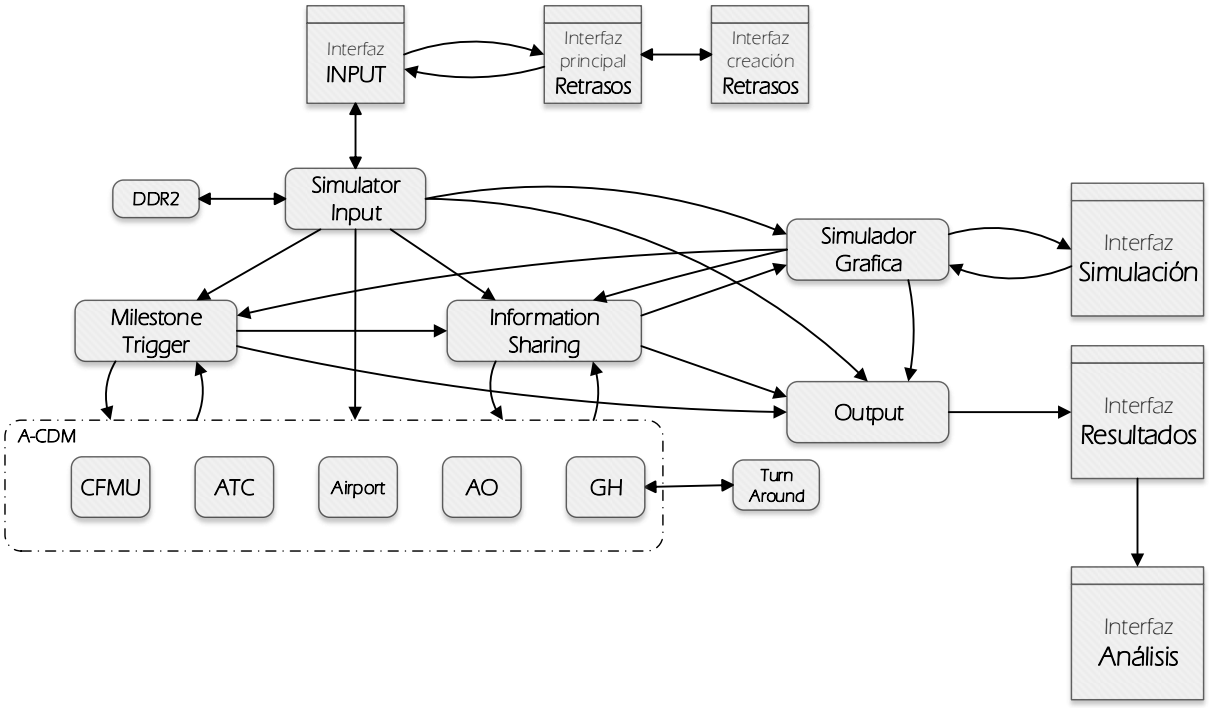


Figura B-3: Diagrama de Comunicaciones con Interfaces

Una vez más no se hará la explicación de cada una de las comunicaciones debido a que en el apartado de explicación de los agentes ya están explicadas con suficiente detalle.

Es necesario comentar que el modelo de la figura B-3 muestra la relación entre los agentes, que agentes intercambian información. No muestra que mensajes intercambian o de qué forma lo hacen. Para ver que mensajes se intercambian en el programa se ha hecho un apartado concreto en la memoria, donde se explican los tipos de mensajes que se intercambian, sus características y su contenido.
Estos diagramas también tienen la limitación de que solo muestra las relaciones que habrá entre los agentes cómo una sola flecha, cuando en realidad entre dos agentes podrían haber más de una flecha porque entre si se hayan establecido diferentes comunicaciones para diferentes propósitos (por esa razón a este diagrama también se le conoce cómo “Diagrama de conocidos”).
Por eso se puede considerar que este diagrama de comunicaciones es una buena herramienta para dar idea de cómo se estructurarán las comunicaciones del programa, pero no da la suficiente información para conocerlas en profundidad.

Otra manera de ver estos diagramas con más detalle de en qué consisten cada una de estas comunicaciones es observar el diagrama de servicios que hay a continuación. En este diagrama de servicios se especifica en que consiste cada una de las comunicaciones.

B.3.3 Modelo de Servicios

En este apartado se detallan las relaciones entre los agentes. Se puede deducir gracias a la fase de análisis de roles e interacciones y a la asignación de roles a agentes.
Para el orden de numeración de los servicios se ha intentado listar todos los servicios de cada agente, apareciendo estos agentes en orden en el que aparecen en la simulación.

Una vez hecho este análisis, podemos definir la lista de servicios cómo la siguiente:

(1) Definir Parámetros Iniciales
(2) Cargar archivos Input
(3) Broadcast parámetros iniciales
(4) Generar archivo input Tráfico Aéreo
(5) Cargar retrasos
(6) Definir Retrasos simulación
(7) Crear retrasos
(8) Broadcast archivos input simulación generados
(9) Broadcast Retrasos
(10) Cargar Resultados para análisis
(11) Iniciar simulación
(12) Broadcast información Avión
(13) Mostrar Simulación en directo
(14) Guardar Simulación
(15) Trigger Milestones
(16) Reloj simulación / Retrasos 1 y 4
(17) Finalizar Simulación
(18) Publicar Actualización Avión
(19) Definir/Actualizar Milestones
(20) Ejecutar retrasos 2 y 3
(21) Calculo tiempos Turn Around
(22) Iniciar CDM para un avión
(23) Cargar Resultados Comparación

Podemos apreciar cómo en la lista de servicios hay un menor número que en la lista de Roles. Esto es debido a que, por definición, los servicios se entienden cómo acciones o roles que lleva a cabo un agente e implican una relación directa con otro agente; en todos los servicios tiene que haber una comunicación entre agentes (o entre agente e interfaz).

Por tanto, todas las acciones que realiza el programa que implican que un agente trabaje solo no están reflejadas en este modelo; por ejemplo, actualizar las tablas internas de los agentes CDM o todas las acciones relacionadas con el generar y analizar los documentos de resultados.

A continuación encontramos el diagrama de servicios resultado de este análisis, figura B-4

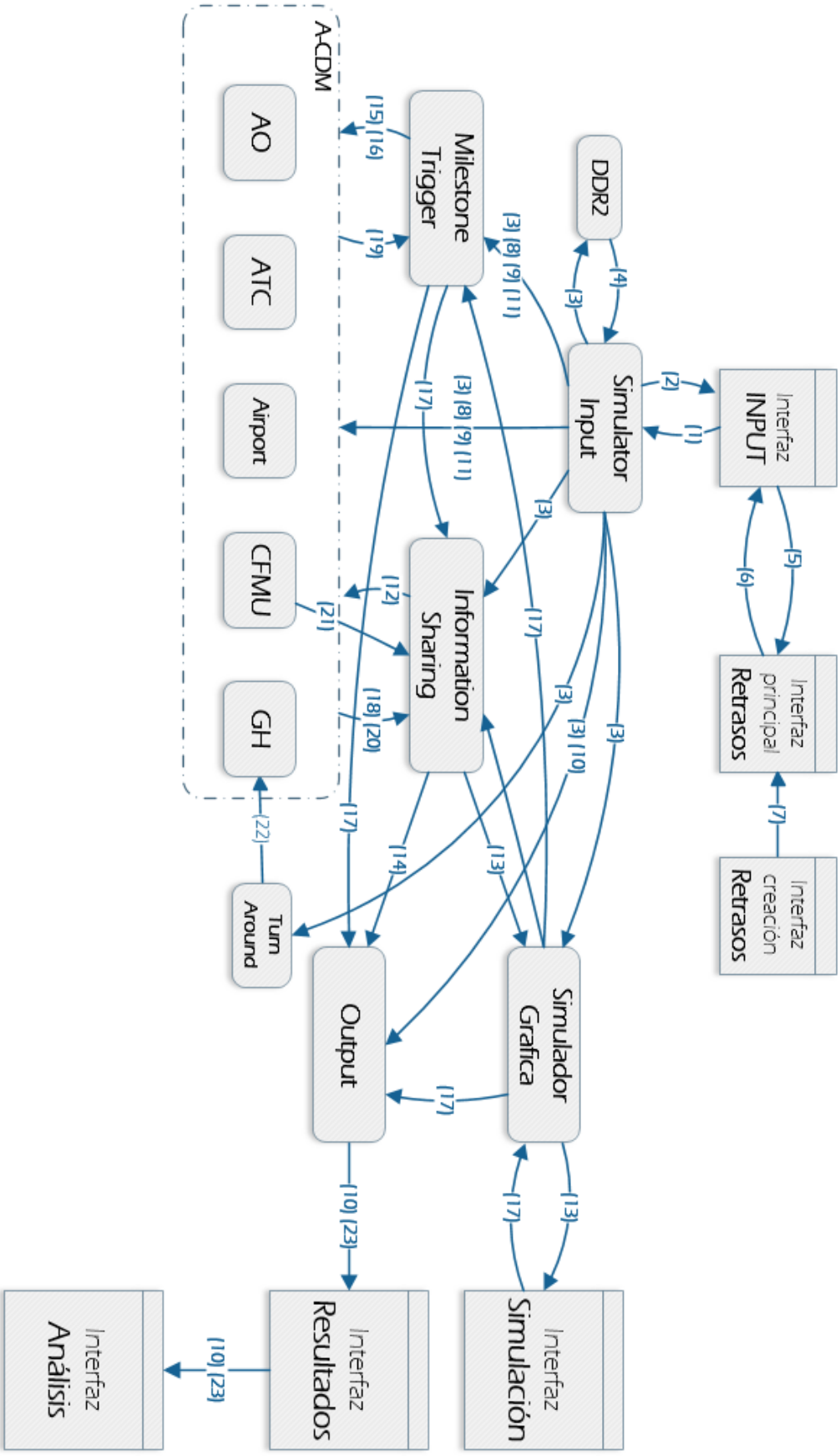


Figura B-4: Diagrama de Servicios

APÉNDICE C. Obtención Datos DDR2 Eurocontrol

El objetivo de este programa es conseguir que simule un tráfico aéreo real, por este motivo se ha encontrado método de conseguir un documento que facilite el tráfico aéreo real a lo largo de un día. Esta información se puede encontrar desde la página de Eurocontrol. A continuación explicaremos detalladamente cómo conseguir descargar este documento input.

El documento Input que añade el tráfico aéreo del día al simulador se obtiene de la base de datos oficial de Eurocontrol, llamada DDR2 (Demand Data Repository). Estos datos se encuentran gratis para investigadores y estudiantes a través de la página web de Eurocontrol. Con el bien de especificar cómo conseguir los datos y que estos tengan las características apropiadas para que el simulador pueda trabajar con ellos, a continuación se puede encontrar una guía de cómo manejar la web de Eurocontrol para descargarlos.

C.1 Acceso a DDR2 Eurocontrol

Es importante saber que para obtener estos datos es necesario estar registrado en la página web de Eurocontrol, concretamente en el apartado de “OneSky Online Extranet”.

https://extranet.eurocontrol.int/

Una vez registrados, para obtener el servicio de los datos del DDR2 es necesario suscribirse a él. Para ello entramos en la pestaña “*Subscrive for online services”* y seleccionamos suscribirnos a “*DDR2 – Demand Data Repository*”. Es posible que esta suscripción no sea aceptada hasta después de unos días, hasta que no sea aceptada no podremos acceder al servicio y descargar los datos. Una vez nos avisen por correo y tengamos el acceso autorizado, entramos otra vez al apartado de “OneSky Online Extranet” con nuestro UserId, y en la pantalla de inicio nos aparecerá el siguiente botón: “DDR2- Demand Data Repository”, tal y cómo podemos en la figura C-1:

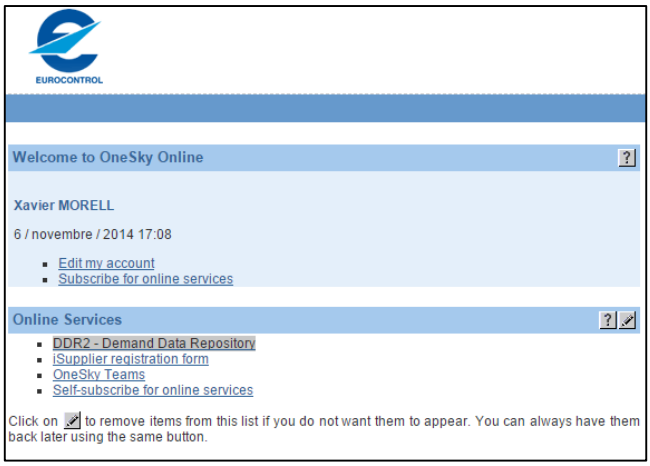


Figura C-1: Fragmento página OneSky Extranet

A partir de este momento ya tenemos acceso a la base de datos de tráfico aéreo de Eurocontrol, y podremos acceder a la información que queremos. Concretamente a los datos del tráfico aéreo de un día concreto en el espacio aéreo europeo.

C.2 Descarga del documento

Al entrar en "DDR2 – Demand Data Repository" se nos abre un menú que nos permite hacer las consultas que queramos, figura C-2.

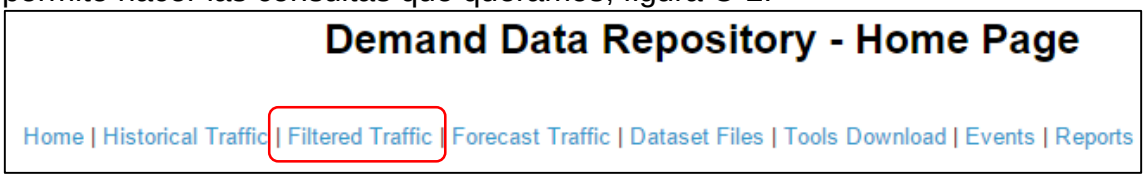


Figura C-2: Encabezado página Demand Data Repository

Para obtener los datos necesarios para el simulador, entramos en el apartado "Filtered Traffic".

Desde este apartado descargaremos el documento de tráfico aéreo que usaremos. En la izquierda encontraremos un formulario (Figura C-3), donde decidiremos los parámetros del documento que queremos obtener de la base de datos. El programa está preparado para trabajar con ficheros de datos del tipo SO6, los cuales muestran la información de los vuelos en espacio aéreo europeo dentro de la franja horaria seleccionada.

El programa está preparado para simular todo el tráfico aéreo a lo largo de un día. Podemos seleccionar fracciones de un día y decidir que solo queremos estudiar unas pocas horas. Pero en ningún caso debemos intentar introducir un documento de un periodo de tiempo comprendido entre 2 días distintos.

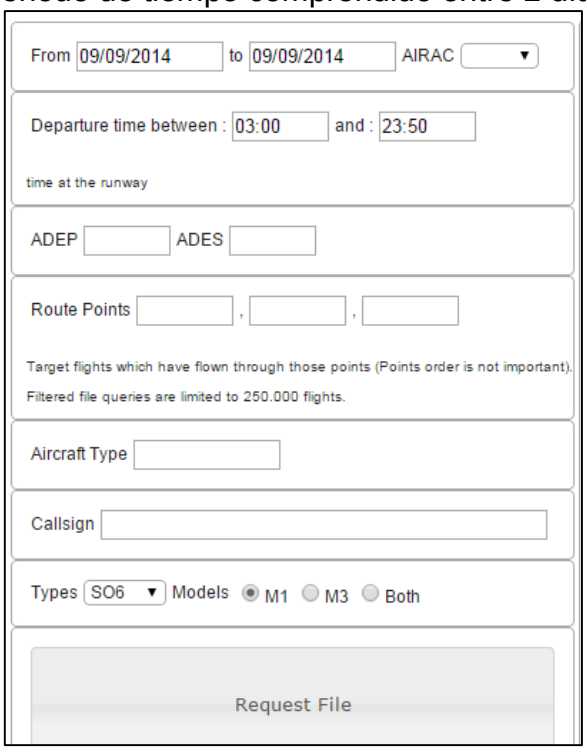


Figura C-3: Formulario Descarga

El programa esta optimizado para poder soportar el archivo de todo el tráfico aéreo en Europa durante un día. No es necesario hacer ningún filtro de restricción de aeropuerto, route points o tipo de aeronave.

La única restricción que hay que tener en cuenta a la hora de obtener un archivo de tráfico aéreo que el programa pueda simular correctamente sería que el tráfico tiene que estar comprendido dentro de un mismo día.

Para obtener hay que conocer los diferentes parámetros de la consulta y cómo utilizarlos.

Al escoger la franja horaria en la que basaremos el estudio estamos decidiendo estudiar todos los aviones que han despegado de algún aeropuerto europeo entre esas hora. Los aviones que en la hora de inicio ya están volando no se tendrán en cuenta, aunque aterricen dentro de la franja horaria seleccionada. De la misma manera, un avión que despegue a las 23:40 sí que aparecerá por mucho que aterrice fuera de la franja de tiempo. Pudiéndose dar el caso de que llegue al aeropuerto de destino pasadas las 24:00 y por lo tanto en un día distinto al del estudio. El programa ya está preparado para estas situaciones.

En el apartado de ADEP y ADES puedes filtrar de inicio para encontrar todos los vuelos con un aeropuerto de salida concreto o uno de llegada. En general el objetivo será simular todo el tráfico aéreo en un aeropuerto, por lo que se recomienda dejar este apartado en blanco. Dentro del programa se puede seleccionar el aeropuerto que se desea simular, por lo que no haría falta descargar este documento más de una vez si queremos hacer diferentes simulaciones para diferentes aeropuertos.

Los textbox del apartado de Route Points, están preparados para entrar el nombre de Waypoints concretos y que la base de datos filtre los vuelos que pasan por esos WayPoints, eliminando los demás.
Para obtener datos de aviones de un modelo concretos se utiliza el Aircraft Type, y en caso de que queramos ver solo un avión, escribiremos el Callsign de dicho avión.

Por lo general estos 3 campos (*Route Points*, *Aircraft Type* y *Callsign*) quedaran vacíos, pues queremos hacer un estudio de todo el tráfico aéreo que circulará por un aeropuerto, no de una fracción de este.

En Types dejaremos siempre seleccionado *SO6* con modelo *M1*, ya viene por defecto con esta selección.

Una vez tengamos los parámetros seleccionados, pediremos el archivo (Request File).

Entonces veremos cómo empieza a descargar-se el documento, aparecerá una barra en medio de la pantalla, cómo la de la figura C-4, indicando que se está preparando el archivo.

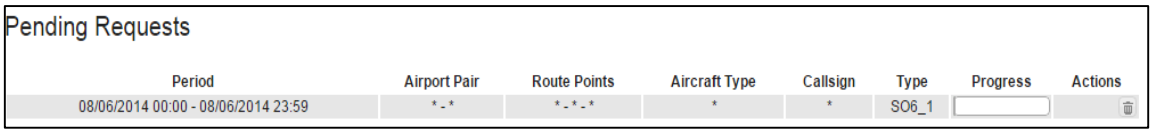


Figura C-4: Pending Requests

Pasados unos segundos, el fichero estará a punto para ser descargado, y aparecerá en la lista de Finished Requests, figura C-5.

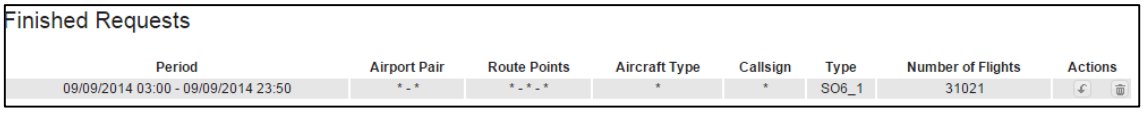


Figura C-5: Finished Requests

Una vez aparezca en la lista de Finished Requests, le damos en el símbolo de descargar en el apartado de Actions. A continuación se descargara el documento SO6. Este documento estará escrito en formato .txt o .so6, en cualquier caso el programa podrá leerlo sin problemas, pero el usuario tendrá de avisar al programa de cuál es el formato a la hora de entrar la ruta del documento.

Podremos cambiarle el nombre al documento, el único requisito para que funcione es referenciar ese nombre correctamente en el simulador. A la hora de empezar la simulación, es importante que este archivo de entrada esté en la misma carpeta que los otros archivos de entrada cómo el Airport.txt.

C.3 Parámetros del documento

Si abrimos el documento de tráfico aéreo descargado de la web de Eurocontrol observaremos que este contiene muchos datos. Cada línea del documento contiene información sobre el recorrido de un avión en cada uno de los segmentos en los que se fracciona su vuelo completo.

La información contenida en cada línea está estructurada tal y cómo muestra la tabla C-1:

#	Información	Tipo	Tamaño	Comentarios
1	Identificador de segmento	char		Inicio segmento "_" final segmento
2	Inicio del vuelo	char	4	Código ICAO
3	Final del vuelo	char	4	Código ICAO
4	Modelo avión	char	4	
5	Hora inicio segmento	num	6	HHMMSS
6	Hora final segmento	num	6	HHMMSS
7	FL inicio segmento	num	1-3	
8	FL final segmento	num	1-3	
9	Estado	num	1	0=subiendo, 1 bajando, 2 crucero
10	Callsign	char		
11	Data inicio segmento	num	6	YYMMDD

12	Data final segmento	num	6	YYMMDD
13	Latitud inicio segmento	num		Minutos y decimales
14	Longitud inicio segmento	num		Minutos y decimales
15	Latitud final segmento	num		Minutos y decimales
16	Longitud final segmento	num		Minutos y decimales
17	Identificador de vuelo	num		Numero único para cada vuelo
18	Secuencia	num		Empieza en 1 y aumenta a cada segmento
19	Longitud del segmento	num		En millas náuticas
20	Color del segmento	num		

Tabla C-1: Documento Tráfico Aéreo Eurocontrol

En el documento extraído de Eurocontrol cada línea contiene la información sobre un segmento de un vuelo. Las líneas consecutivas pertenecen a segmentos consecutivos de un mismo avión. De esta manera podemos saber que cuando el valor de *Secuencia* sea 1, significara que estamos tratando con un vuelo nuevo, y en las líneas consecutivas ira añadiendo valor a la secuencia 1, 2, 3, 4, etc. hasta que se terminen los segmentos del vuelo de ese avión, en ese momento volveremos a empezar la secuencia en 1, dando información de un nuevo vuelo.

APÉNDICE D. Manual del Software

En este apéndice se explicará todo lo que un usuario debe conocer para poder ejecutar el programa correctamente.

Nos centraremos en cómo preparar el programa para que sea posible ejecutarlo desde Eclipse. También comentaremos rápidamente los documentos necesarios para realizar una simulación.

Al realizar una simulación, también encontramos una serie de parámetros que se pueden escoger pero que no aparecen en ninguna interfaz. También se comentará cuáles son estos parámetros, que representan, donde encontrarlos y cómo modificarlos.

D.1 Cómo obtener el código

El código del programa se puede conseguir descargándolo de la web Github a través del siguiente link:

https://github.com/A-CDMteam/SimuladorV2.0

En este link podemos obtener el código del programa, con una versión de ejemplo de Airport.txt y Tur-AroundTimer.txt.

D.2 Cómo ejecutarlo (Run configurations)

Al tratarse de un programa en JADE, y teniendo en cuenta que es un programa descentralizado donde se ejecutan una serie de agentes, la manera de ejecutarlo es diferente a la habitual de los programas en Java.
Por este motivo es necesario definir las *Run configurations*, donde definiremos los agentes que se ejecutarán.

Para hacer esta configuración es necesario ir a la opción de *Run Configurations* de Eclipse. Se abrirá una ventana cómo en la figura D-1, donde escribiremos las configuraciones de la simulación.

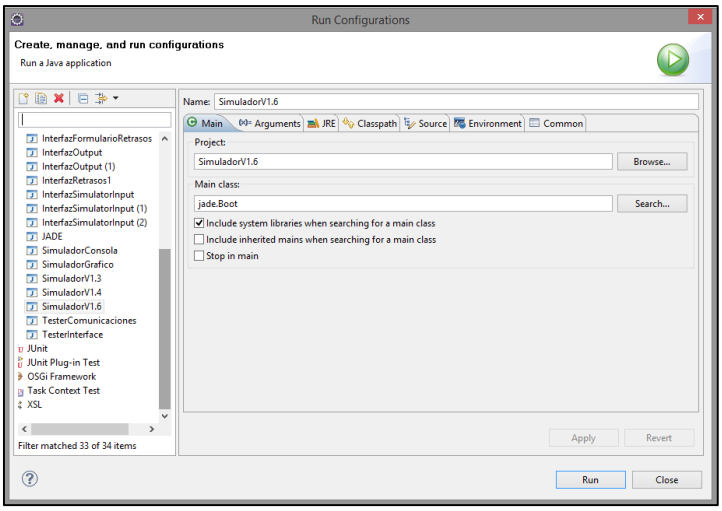


Figura D-1: Run Configurations

Primero nos fijaremos en el lado izquierdo donde aparece una lista de configuraciones, por lo que si queremos modificar una configuración ya creada, la seleccionaremos y la modificaremos a nuestro gusto. Si queremos crear una nueva configuración pulsaremos el botón de "*New launch configuration"*.

Al pulsar este botón estaremos creando una configuración nueva. Si estamos creando una configuración nueva, lo primero que tendremos de hacer es escoger el nombre:

En la opción *Name* escribiremos el nombre que le daremos a las configuraciones que procederemos a escribir o modificar

En la ventana que se nos ha abierto de *Run configurations* tenemos una serie de pestañas tal y cómo podemos observar con más detalle en la figura D-2:

Figura D-2: Pestañas Run Configurations

De esta lista de pestañas solo nos interesará modificar *Main* (figura D-3) y *Arguments* (figura D-4). En estas dos pestañas se tienen que realizar las siguientes modificaciones:

Main:

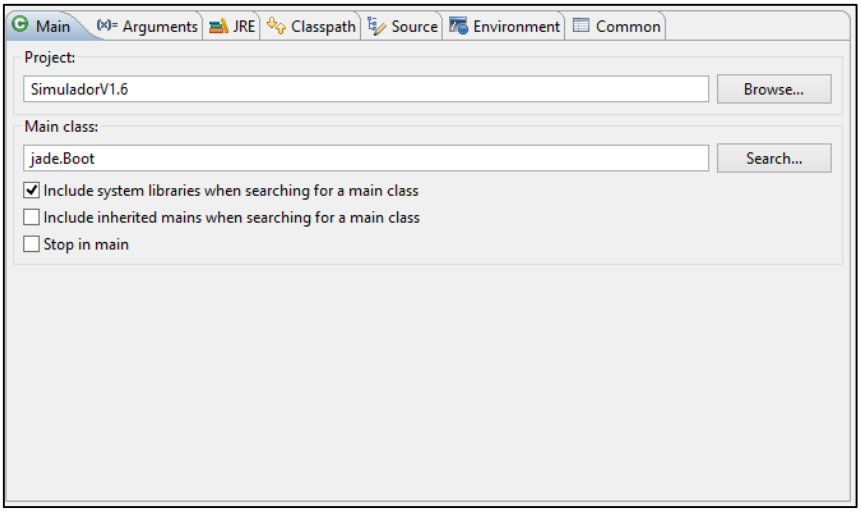


Figura D-3: Pestaña Main (Run Configurations)

- Project: En este apartado escribiremos el nombre del proyecto.
- Main Class: Al tratarse de una simulación con Jade escribiremos siempre *jade.Boot*.

Este *jade.Boot* es la clase por defecto que utiliza Jade para crear el contenedor y añadir e iniciar los agentes de la simulación.

Luego siempre seleccionaremos la casilla de *Include system libraries when searching for a main class.* Las otras opciones no serán seleccionadas.

Arguments

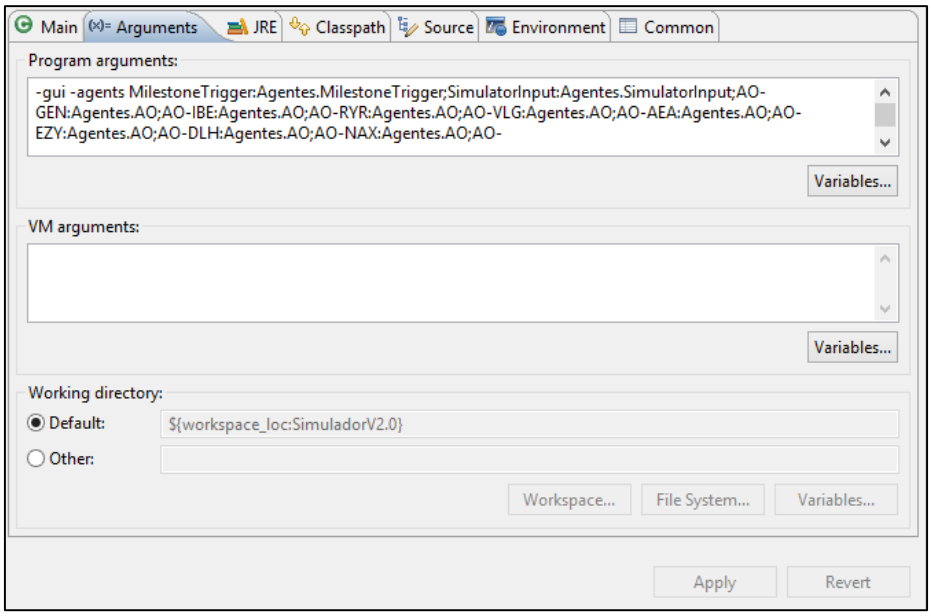


Figura D-4: Arguments (Run Configurations)

En la pestaña *Arguments* únicamente tendremos de modificar el apartado de *Program Arguments*. En esta casilla escribiremos todos los agentes que tendremos en funcionamiento durante la simulación. Como el programa tiene una serie de agentes fijos siempre escribiremos lo siguiente:

```
-agents
MilestoneTrigger:Agentes.MilestoneTrigger;SimulatorInput:Agentes.SimulatorInput;AO-
GEN:Agentes.AO;CFMU:Agentes.CFMU;InfoSharing:Agentes.InfoSharing;ATC:Agentes.ATC;Airport:Agentes.Airport;Interfaz:Agentes.SimuladorGrafica;DDR2:Agentes.DDR2;GH-
GEN:Agentes.GH;TurnAround:Agentes.TurnAround;Output:Agentes.Output;
```

Cada uno de los agentes utilizados durante la simulación se definirá con la siguiente estructura:

NombreAgente:Agentes.NombreArchivoAgente

De manera que copiando esa línea en *Program Arguments* estaremos creando los agentes de: SimulatorInput, AO-GEN, CFMU, InfoSharing, ATC, Airport, Interfaz (SimuladorGrafica). DDR2, GH-GEN, TurnAround y Output.

Si queremos añadir aerolíneas y empresas de GH con agente propio se tendrá que modificar el texto entrado en *Program Arguments* de la siguiente manera: Escribiremos para cada aerolínea extra que queramos añadir lo siguiente:

```
AO-RYR:Agentes.AO;
```

Donde “RYR” son las siglas de la aerolínea que queremos añadir en este caso. Es muy importante tener en cuenta que si se añade una aerolínea en las *Run Configurations* está también tiene de aparecer en el documento de Airport.

Y para cada compañía de ground handling extra añadiremos lo siguiente:

```
GH-FrankfurtAGS:Agentes.GH;
```

Donde “FrankfurtAGS” es el nombre de la compañía que queremos añadir en este caso. Es muy importante tener en cuenta que si se añade una compañía de ground handling en las *Run Configurations* está también tiene de aparecer en el documento de Airport.txt

Si nos fijamos en la figura anterior podemos observar que justo antes de poner la lista de agentes que aparecerán en el *Program Arguments* aparece el texto

```
-gui
```

Poner el `-gui` no afecta al funcionamiento del programa, pero es aconsejable añadirlo, nos abrirá la ventana de *JADE Remote Agent Management GUI.* Esta interfaz de JADE es muy útil para ver los agentes que se están ejecutando y para capturar las comunicaciones del programa. En el apéndice “Tutorial JADE” se encuentra una descripción de esta interfaz, figura D-5 y de cómo utilizarla.

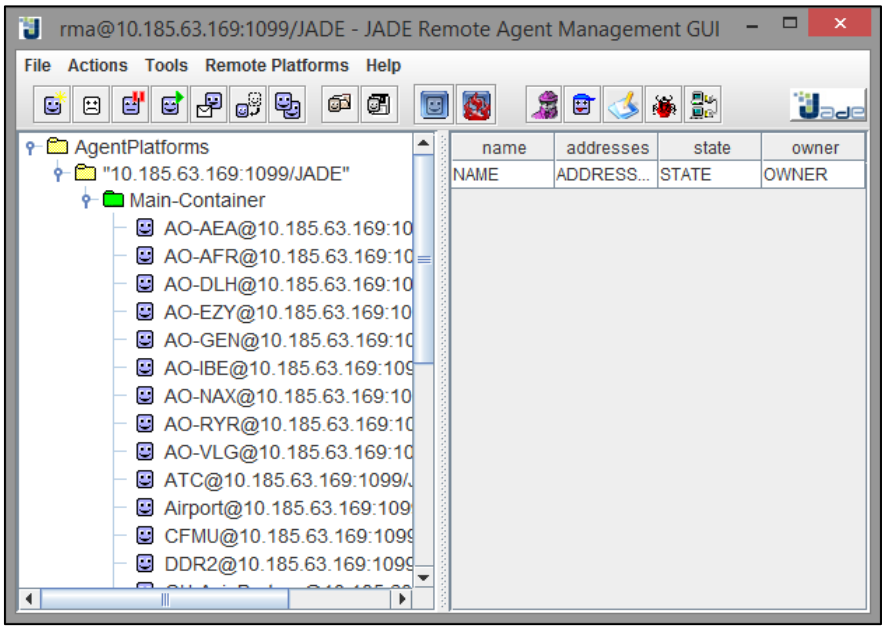


Figura D-5: JADE Remote Agent Management GUI

En las últimas simulaciones realizadas, se ha utilizado el siguiente *Program Arguments*

```
-gui -agents
MilestoneTrigger:Agentes.MilestoneTrigger;SimulatorInput:Ag
entes.SimulatorInput;AO-GEN:Agentes.AO;AO-
IBE:Agentes.AO;AO-RYR:Agentes.AO;AO-VLG:Agentes.AO;AO-
AEA:Agentes.AO;AO-EZY:Agentes.AO;AO-DLH:Agentes.AO;AO-
NAX:Agentes.AO;AO-
AFR:Agentes.AO;CFMU:Agentes.CFMU;InfoSharing:Agentes.InfoSh
aring;ATC:Agentes.ATC;Airport:Agentes.Airport;Interfaz:Agen
tes.SimuladorGrafica;DDR2:Agentes.DDR2;GH-
GEN:Agentes.GH;GH-IBEServices:Agentes.GH;GH-
Lesma:Agentes.GH;GH-FrankfurtAGS:Agentes.GH;GH-
AviaPartner:Agentes.GH;GH-
Swissport:Agentes.GH;TurnAround:Agentes.TurnAround;Output:A
gentes.Output;
```

Una vez preparado el *Program Arguments* ya tenemos las *Run Configurations* preparadas para realizar una simulación.

D.3 Inputs necesarios (documentos)

Una vez preparadas las *Run Configurations* para empezar correctamente una simulación solo falta asegurar que tenemos los documentos de entrada preparados. En el apartado 4.5.1 de la memoria podemos encontrar detalladamente la información sobre cómo tienen de estar escritos los documentos de entrada necesarios.
A continuación explicaremos brevemente estos documentos con tal de poder mostrar las principales características de estos documentos:

- Documento DDR2: Este documento se descarga directamente de la página web de Eurocontrol (Mirar Apéndice Obtención Datos DDR2 Eurocontrol). Contiene los datos del tráfico aéreo que se simulará. A la hora de descargarlo puede estar en formato .txt o en .so6; se recomienda comprobarlo, ya que a la hora de definirlo en el programa es necesario escribir bien la extensión del documento para que el programa lo encuentre.

- Airport.txt: El documento Airport.txt es el que contiene toda la información relacionada con el aeropuerto. A continuación en la tabla D-1 encontramos un ejemplo de la información que contiene este documento.

Parámetro documento	**Significado**
LEBL	Código ICAO del aeropuerto a estudiar
RYR IBE VLG …	Siglas de las aerolíneas con agente propio, tienen de coincidir con las que aparecen en la Run Configuration
Op/h-IN 30	Operaciones/hora de la pista de aterrizaje, no puede ser superior a 30 ni inferior a 1
OP/h-Out 30	Operaciones/hora de la pista de salidas, no puede ser superior a 30 ni inferior a 1

A380-1 B744-2 A346-2	Modelos de avión y categoría de cada uno, los modelos que no salgan en la lista se considerarán de categoría 4 siempre.
FIR1 FIR2 FIR3 FIR4 FIR5	Waypoints de la FIR del aeropuerto
APP1 APP2 APP3 APP4 APP5	Waypoints de los puntos de inició de la fase de aproximación al aeropuerto
IBEServices-IBE Lesma-RYR	Compañías de Ground Handling con las siglas de las aerolíneas con las que tiene contrato cada una.

Tabla D-1: Parámetros documento Airport

Una manera rápida de obtener este documento en formato correcto es coger el documento de ParametrosSimulacion.txt de otra simulación y borrar las líneas previas al aeropuerto. De esta manera obtendremos unos parámetros iniciales iguales que a esa simulación.

Un ejemplo de este documento es el de la figura D-6:

```
LEBL
RYR  IBE  VLG  AEA  EZY  DLH  NAX  AFR
Op/h-In   30
Op/h-Out  30
A380-1 B744-2 A346-2 B773-2 A343-2 B772-2 A333-2 A332-2
B763-2 A306-2 B747-2 B767-2 B777-2 A300-2 A310-2 A330-2
A340-2 MD11-2 DC10-2 A319-3 A320-3 A321-3 B752-3 B757-3
B737-3 MD80-3 DC9-3 E190-3 B722-3 B739-3 B738-3 MD83-3
B737-3 A318-3 MD82-3 B736-3 B735-3 B734-3 B733-3 DC95-3
B717-3 E190-3 DC93-3 F100-3 RJ100-3 B463-3 RJ85-3 B462-3
CRJ9-3 F70-3 E170-3 CRJ7-3 GLF4-3 DH8D-3 E45X-3 CRJ2-3
CRJ1-3 AT72-3 F50-3 DH8C-3 E145-3 E135-3 AT45-3 DH8B-3
DH8A-3 AT43-3 SF34-3 E120-3 H25B-3 C650-3 B190-3
FIR1 FIR2 FIR3 FIR4 FIR5
APP1 APP2 APP3 APP4 APP5
IBEServices-IBE     Lesma-RYR FrankfurtAGS-TAP-DUB-VLG
     AviaPartner-AEA-NAX-AFR-MON   Swissport-SWR-WZZ
```

Figura D-6: Ejemplo Airport.txt

- Retrasos.txt: Este documento se genera de forma automática a través de definir los retrasos de la simulación con la Interfaz Retrasos. Esta interfaz genera y guarda este documento, para que en otra simulación pueda ser cargado y se utilicen los mismos retrasos.

- Turn-Around-Time.txt: Este documento contiene la base de datos de tiempos de Turn Around por modelo y Aerolínea que se utiliza en el programa. Si hay problemas a la hora de leer y cargar este documento, el programa utiliza los tiempos por defecto en el programa.
Este documento puede ser modificado fácilmente añadiendo más aviones y aerolíneas o modificando cualquiera de los tiempos que vienen en el documento inicial.

D.4 Manual de Usuario

En este apartado comentaremos brevemente las interfaces que el usuario encontrará al ejecutar el simulador, y todas las acciones que puede realizar desde cada una de estas interfaces.

El programa está dotado de un total de 7 interfaces:

- 1 Interfaz de inicio
- 3 interfaces para la administración de los retrasos
- 2 Interfaces para mostrar los resultados analizados
- 1 Interfaz para analizar los resultados

A continuación mostraremos cada una de las interfaces y explicaremos las acciones que podemos realizar desde cada uno de estos:

D.4.1 Interfaz A-CDM Simulator Input

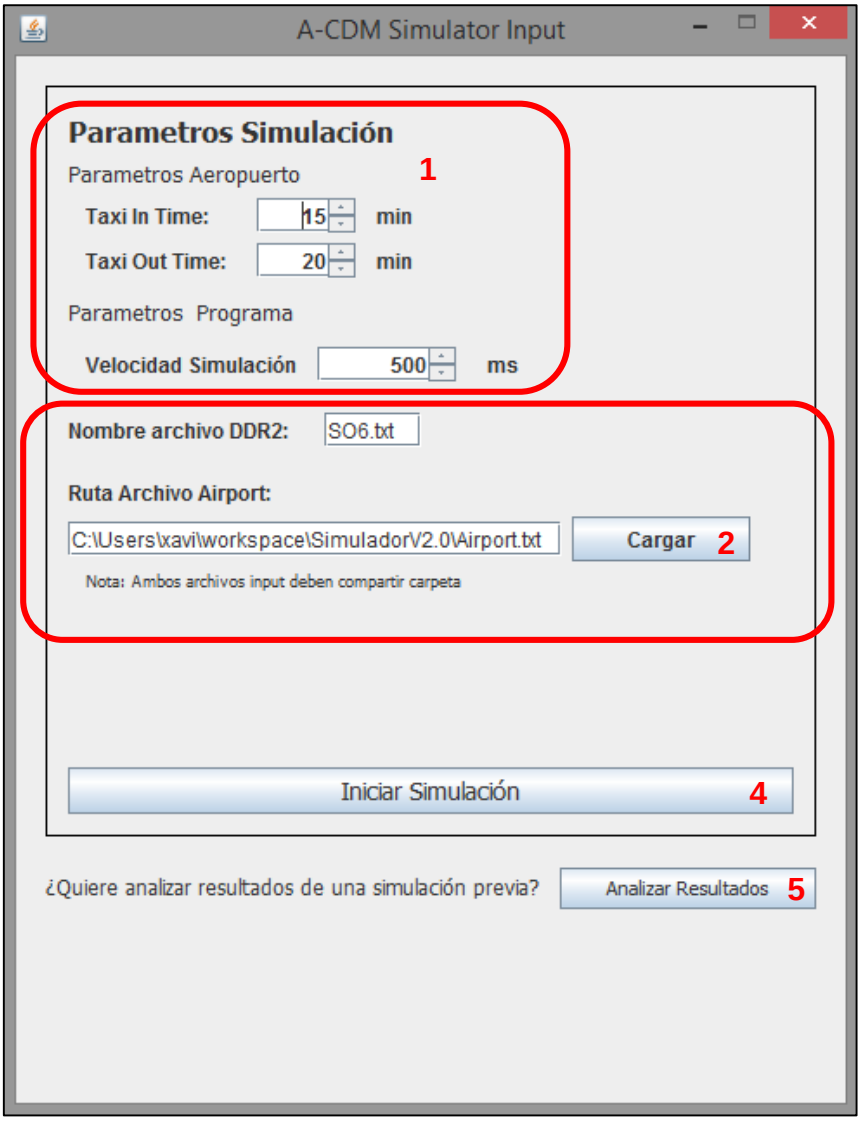


Figura D-7: Interfaz A-CDM Simulator Input

- **1**: Selección de parámetros de simulación.
- **2**: Indicamos los nombres de los documentos input que utilizaremos para la simulación. Al pulsar el botón Cargar el programa leerá esos documentos y empezará a realizar los cálculos. Al pulsar este botón en la interfaz aparecerá un nuevo botón, figura D-8:

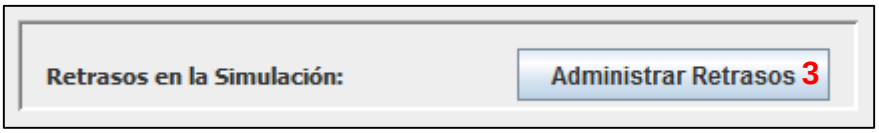


Figura D-8: A-CDM Simulator Input (Retrasos)

- **3**: Botón para administrar los retrasos que se aplicarán durante la simulación. Si seleccionamos este botón podemos definir los retrasos que se aplicarán a lo largo de la simulación. Si no queremos retrasos ignoraremos este botón. Este botón abrirá la interfaz ***Retrasos Simulación.***
- **4**: Botón para iniciar la simulación. Si no se han cargado los documentos input (2), este botón no funciona. Al seleccionarse esta opción se abrirá la Interfaz ***A-CDM Simulator.***
- **5**: Botón Analizar Resultados: Si no queremos realizar una simulación y solo queremos visualizar los resultados de una simulación ya realizada tendremos de seleccionar este botón. Al pulsar este botón en la interfaz aparecerá un nuevo botón, figura D-9:

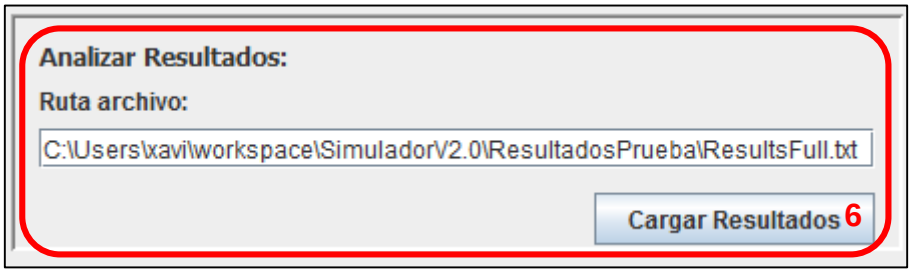


Figura D-9: A-CDM Simulator Input (Cargar Resultados)

- **6**: Analizar Resultados: En este apartado escribiremos la ruta donde encontrar el documento con los resultados calculados por el programa en otra simulación. Al seleccionar esta opción se abrirá la interfaz ***A-CDM Results***

D.4.2 Interfaz Retrasos Simulación

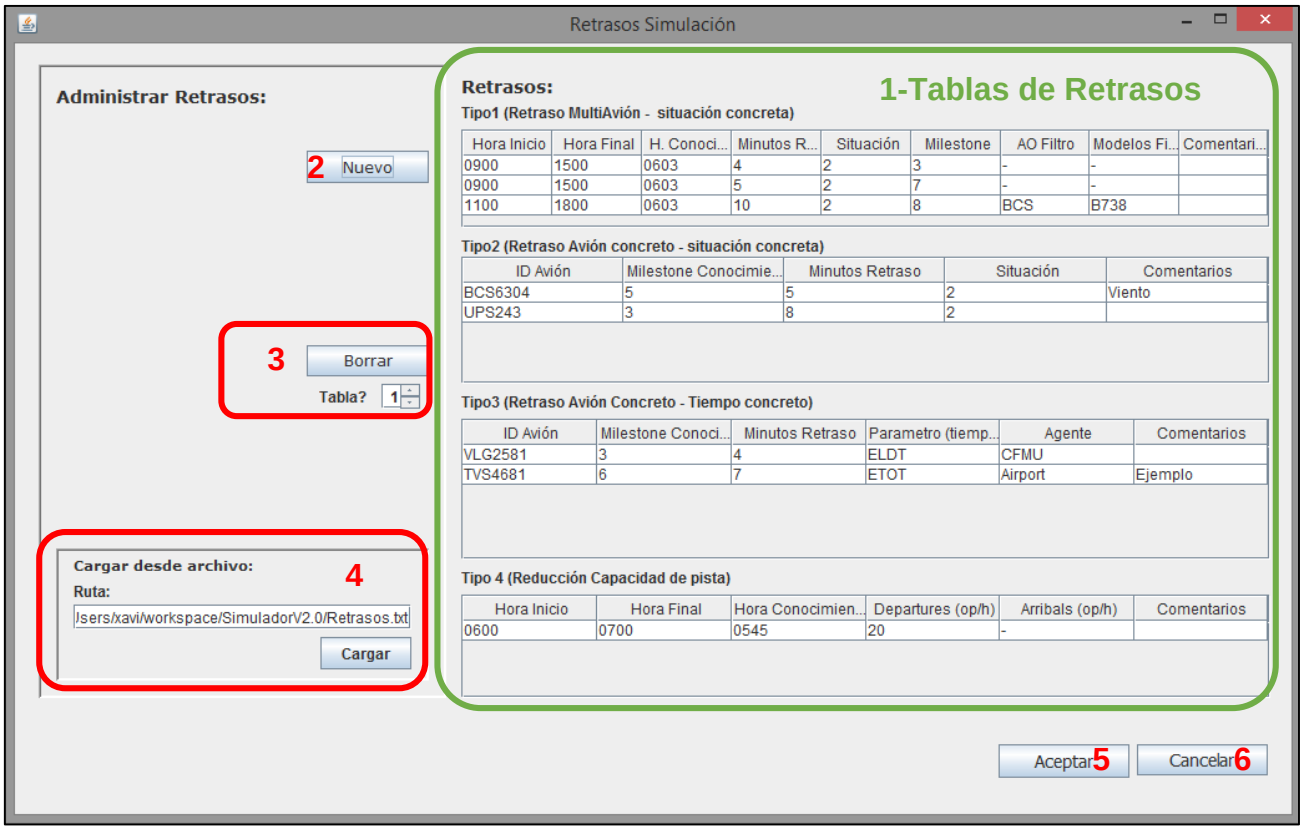


Figura D-10: Interfaz Retrasos Simulación

- **1-Tablas de Retrasos**: Aquí podemos observar las tablas con los retrasos que estamos definiendo y que se aplicarán durante la simulación. Tenemos 4 tablas distintas, cada una informa sobre los retrasos de un tipo.
- **2**: Botón Nuevo, desde aquí accederemos a la zona que nos permite crear un nuevo retraso con los elementos deseados. Al pulsar este botón se abrirá la interfaz ***Formulario Retrasos***
- **3**: Borrar Retraso: En este apartado podemos decidir eliminar un retraso de los ya definidos. Primero usaremos el spinner para seleccionar la tabla de la que retrasaremos, y luego seleccionaremos el retraso que borraremos (pulsaremos en la propia tabla el retraso que queremos eliminar, para dejarlo seleccionado). Al pulsar el botón ese retraso será eliminado.
- **4**: Botón Cargar. Sirve para cargar el documento de retrasos que se encuentra en el TextBox de la Ruta. El programa leerá el documento y añadirá a las tablas los retrasos que estén en el documento. Este botón solo se puede seleccionar una vez por simulación.

- **5**: Aceptar. Este botón es la confirmación de la aplicación de los retrasos definidos en las tablas. Hasta que no se pulse este botón los retrasos no entran al programa y no se aplican. Una vez pulsado este botón volveremos a la interfaz ***A-CDM Guardar Retrasos***
- **6**: Cancelar. Con este botón estamos diciendo al programa que no queremos aplicar ningún retraso. Por lo que todos esos retrasos que se han definido anteriormente serán eliminados. Una vez pulsado este botón volveremos a la interfaz ***A-CDM Simulator Input.***

D.4.3 Interfaz Formulario Retrasos

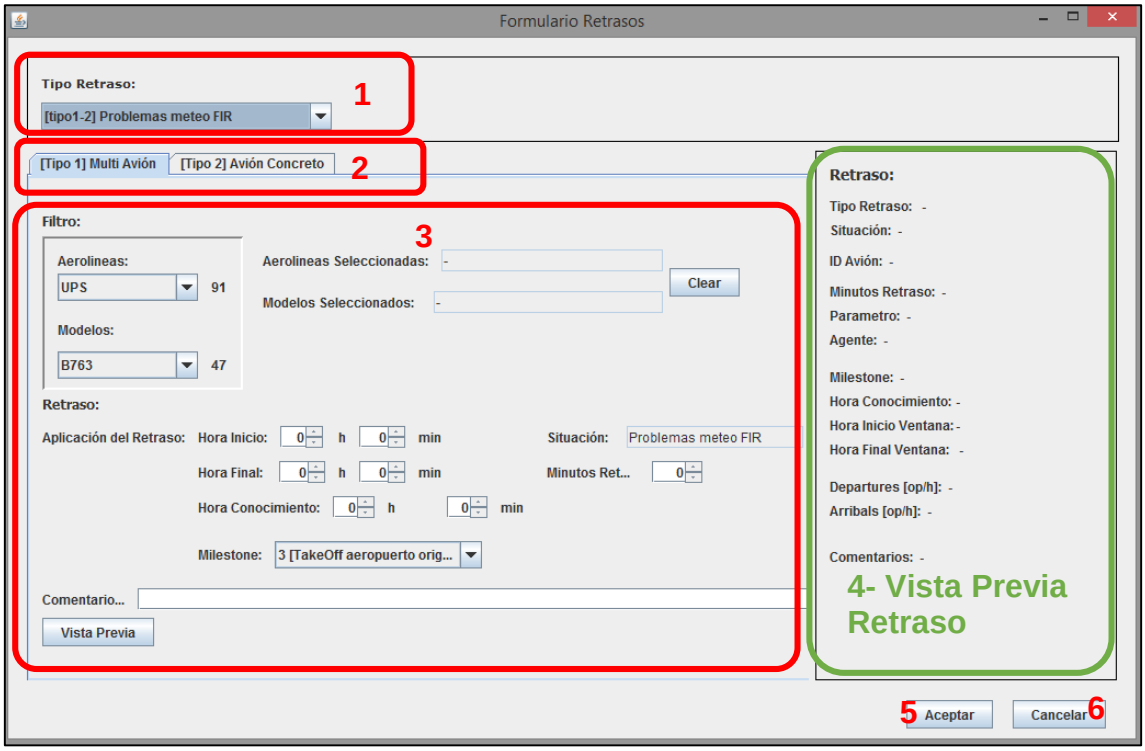


Figura D-11: Interfaz Formulario Retrasos

- **1**: Tipo de Retraso. Aquí seleccionaremos el tipo de retraso que queremos definir. Los botones y las variables que aparecerán en el panel 3 dependerán de que se elija en esta lista.
- **2**: Depende del Tipo de Retraso seleccionado (1), tendremos de realizar otra selección, para terminar de definir cuál es el tipo de retraso que estamos definiendo.
- **3**: Parámetros Retraso. En este apartado definiremos todos los parámetros relacionados con el retraso, aviones afectados, tiempos, cuando resulta afectado, motivos… Cuando terminemos de definir el retraso pulsaremos el botón *Vista Previa.* Según el Tipo de Retraso seleccionado las preguntas para definir el retraso serán unas u otras.

- **4-Vista Previa Retraso**. Aquí podemos ver la información ordenada del retraso que estamos a punto de definir. Esta información variará según la información definida en los Parámetros Retraso en el momento de pulsar el botón *Vista Previa.*
- **5**. Aceptar, al clicar este botón estamos indicando al programa que el retraso definido, según la información de Retraso (4), es correcto. Una vez pulsamos aquí, el programa añadirá este retraso a la lista de retrasos definidos. Volveremos a la interfaz ***Retrasos Simulación.***
- **6**: Cancelar. Si finalmente decidimos no definir un nuevo retraso, podemos pulsar este botón y volveremos a la interfaz ***Retrasos Simulación*** sin definir ningún retraso nuevo.

D.4.4 Interfaz Guardar Retrasos

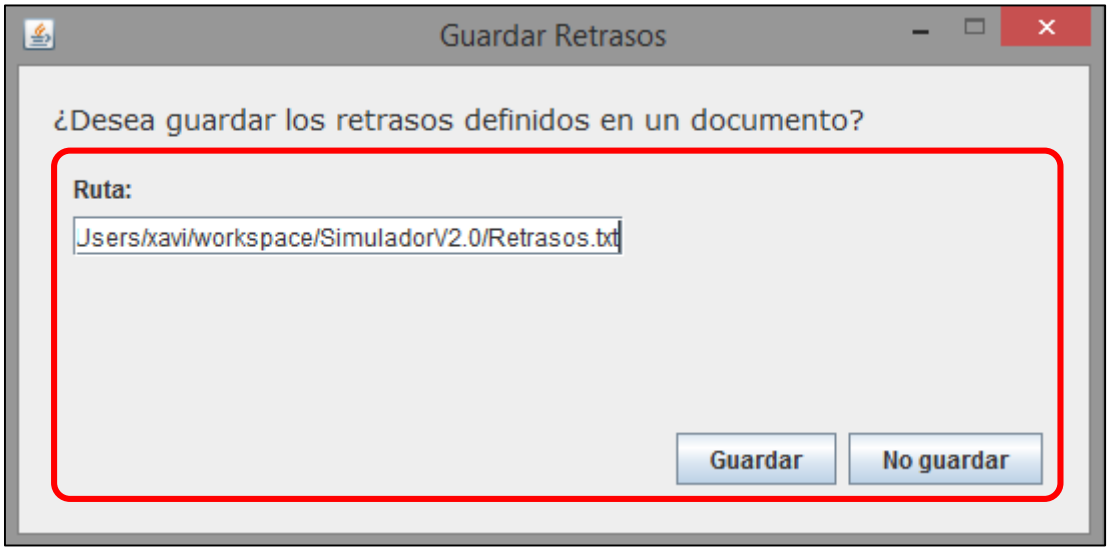


Figura D-12: Interfaz Guardar Retrasos

- **Guardar**: En esta interfaz podemos decidir si queremos guardar una segunda copia de seguridad de los retrasos definidos (una copia siempre se guarda en el workspace). Al pulsar el botón Guarda, el programa guardará esta segunda copia en la carpeta que encuentra en Ruta. Seguidamente volveremos a la interfaz ***A-CDM Simulator Input***.
- **No Guardar**, Si no queremos una segunda copia del documento de retrasos que se aplicarán pulsaremos este botón. Seguidamente volveremos a la interfaz ***A-CDM Simulator Input***.

D.4.5 Interfaz A-CDM Simulator

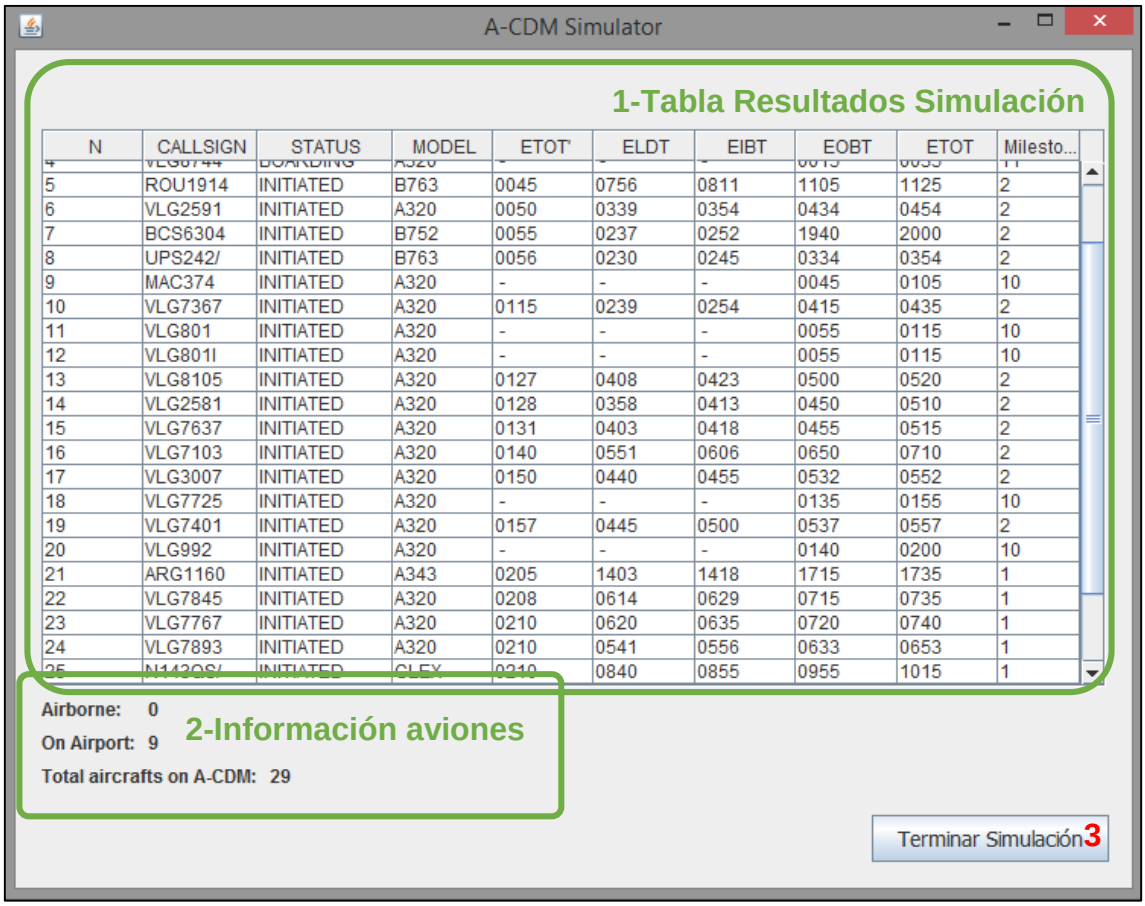


Figura D-13: Interfaz A-CDM Simulator

- **1-Tabla Resultados Simulación**: Esta zona de la interfaz es únicamente informativa. Aquí encontraremos una tabla con la información de los aviones simulados que se actualizará constantemente según los nuevos cálculos del simulador.
- **2-Información aviones**: Encontramos una serie de labels que nos informan sobre los aviones que se encuentran volando hacia el aeropuerto (*Airborne*), dentro del aeropuerto (*On Airport*) y dentro de la simulación(*Total aircrafts on A-CDM*). Este último label va subiendo según los aviones realizan el primer milestone en el programa, pues es en ese momento cuando se definen sus parámetros.
- **3-Terminar Simulación**. El usuario tiene la opción de terminar una simulación en cualquier momento. En el momento que se pulse este botón el programa dejará de calcular nuevos tiempos. La tabla de resultados en el momento de pulsar el botón se definirá cómo los resultados finales de la simulación. Acto seguido se abrirá la interfaz ***A-CDM Resultados.*** Si dejamos al programa simular hasta el último avión, cuando termine todos los cálculos el programa cerrará esta interfaz e igualmente abrirá la

interfaz ***A-CDM Resultados***, pero ahora mostrará los resultados encontrados al final de la simulación.

D.4.6 Interfaz A-CDM Resultados

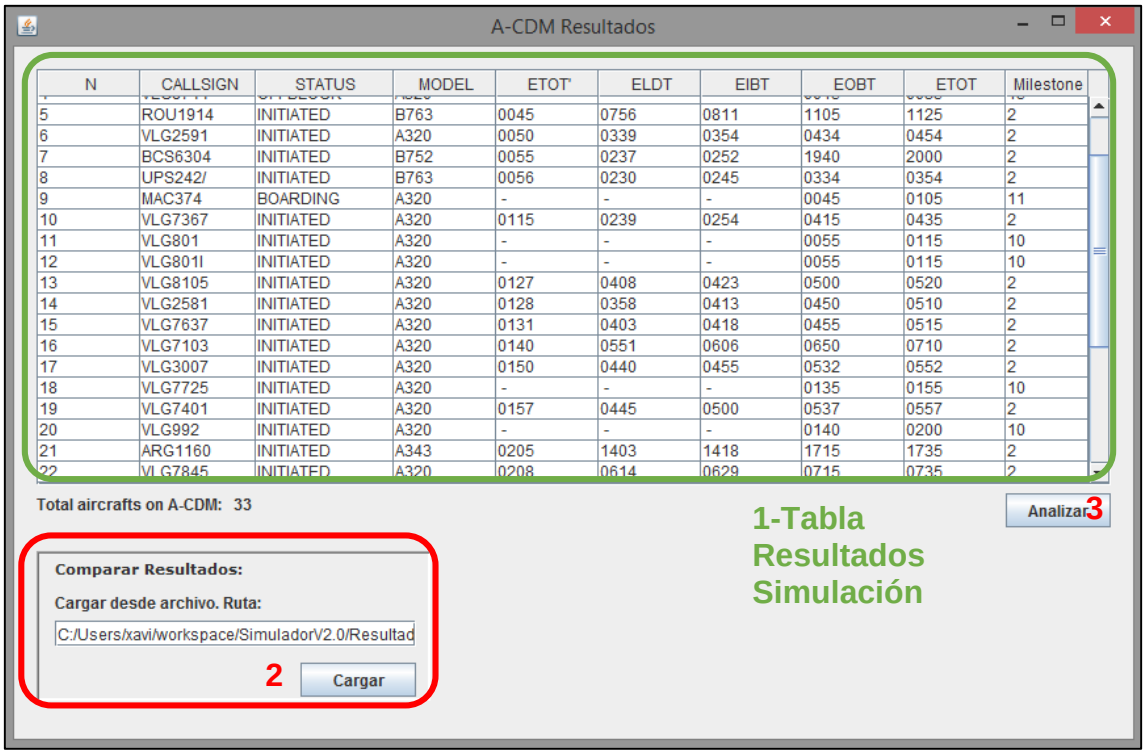
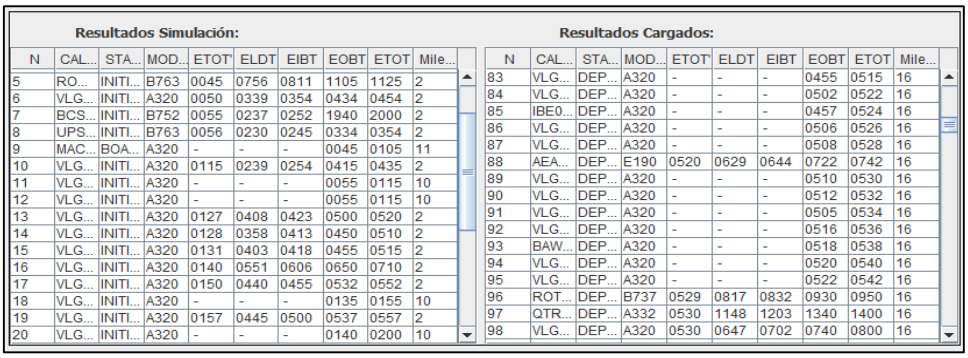


Figura D-14: Interfaz A-CDM Resultados

- **1-Tabla Resultados Simulación**: Esta zona de la interfaz es únicamente informativa. Aquí encontraremos una tabla con la información de los aviones al final de la simulación. Esta tabla se puede llenar con información de 3 maneras distintas:
    - Simulación Finalizada: Se ha dejado al programa realizar una simulación hasta el final.
    - Simulación parcial: Se ha pulsado el botón *Terminar Simulación* en la interfaz *A-CDM Simulator*
    - Resultados simulación documento: Estamos mostrando los resultados de una simulación realizada anteriormente de la cual el programa solo ha leído los resultados encontrados anteriormente y ahora los muestra por pantalla. Esto sucede al pulsar el botón *Cargar Resultados* en la interfaz *A-CDM Simulator Input*.
- **2**: Puede Ser que decidamos cargar una segunda tabla de resultados (esta será siempre de un documento externo) a modo de realizar comparaciones con la tabla que ya tenemos abierta. El programa buscará en la Ruta el documento con los resultados de una simulación anterior y

la cargará. La zona *1-Tabla Resultados Simulación* se verá modificada a la figura D-15:



Resultados Simulación:

N	CAL...	STA...	MOD...	ETOT	ELDT	EIBT	EOBT	ETOT	Mile...
5	RO...	INITI...	B763	0045	0756	0811	1105	1125	2
6	VLG...	INITI...	A320	0050	0339	0354	0434	0454	2
7	BCS...	INITI...	B752	0055	0237	0252	1940	2000	2
8	UPS...	INITI...	B763	0056	0230	0245	0334	0354	2
9	MAC...	BOA...	A320	-	-	-	0045	0105	11
10	VLG...	INITI...	A320	0115	0239	0254	0415	0435	2
11	VLG...	INITI...	A320	-	-	-	0055	0115	10
12	VLG...	INITI...	A320	-	-	-	0055	0115	10
13	VLG...	INITI...	A320	0127	0408	0423	0500	0520	2
14	VLG...	INITI...	A320	0128	0358	0413	0450	0510	2
15	VLG...	INITI...	A320	0131	0403	0418	0455	0515	2
16	VLG...	INITI...	A320	0140	0551	0606	0650	0710	2
17	VLG...	INITI...	A320	0150	0440	0455	0532	0552	2
18	VLG...	INITI...	A320	-	-	-	0135	0155	10
19	VLG...	INITI...	A320	0157	0445	0500	0537	0557	2
20	VLG...	INITI...	A320	-	-	-	0140	0200	10

Resultados Cargados:

N	CAL...	STA...	MOD...	ETOT	ELDT	EIBT	EOBT	ETOT	Mile...
83	VLG...	DEP...	A320	-	-	-	0455	0515	16
84	VLG...	DEP...	A320	-	-	-	0502	0522	16
85	IBE0...	DEP...	A320	-	-	-	0457	0524	16
86	VLG...	DEP...	A320	-	-	-	0506	0526	16
87	VLG...	DEP...	A320	-	-	-	0508	0528	16
88	AEA...	DEP...	E190	0520	0629	0644	0722	0742	16
89	VLG...	DEP...	A320	-	-	-	0510	0530	16
90	VLG...	DEP...	A320	-	-	-	0512	0532	16
91	VLG...	DEP...	A320	-	-	-	0505	0534	16
92	VLG...	DEP...	A320	-	-	-	0516	0536	16
93	BAW...	DEP...	A320	-	-	-	0518	0538	16
94	VLG...	DEP...	A320	-	-	-	0520	0540	16
95	VLG...	DEP...	A320	-	-	-	0522	0542	16
96	ROT...	DEP...	B737	0529	0817	0832	0930	0950	16
97	QTR...	DEP...	A332	0530	1148	1203	1340	1400	16
98	VLG...	DEP...	A320	0530	0647	0702	0740	0800	16

Figura D-15: Tablas Resultados (A-CDM Resultados)

La tabla de la izquierda es la que ya teníamos anteriormente, mientras que en la derecha de la interfaz aparecerá una nueva tabla con los resultados obtenidos en el documento leído.

- **3**: analizar, al pulsar este botón estamos indicando al programa que queremos realizar un análisis un poco más profundo de los datos encontrados. Debemos ser conscientes que el análisis se realizará de las tablas que se estén mostrando por pantalla en el momento de pulsar el botón. Si solo vemos una tabla, el análisis se realizará únicamente de esa tabla. Y si tenemos otra tabla de un documento cargado, realizaremos un análisis de las 2 tablas a la vez, facilitando la comparación de los 2 resultados distintos. Se abrirá la interfaz ***A-CDM Análisis.***

D.4.7 Interfaz A-CDM Análisis

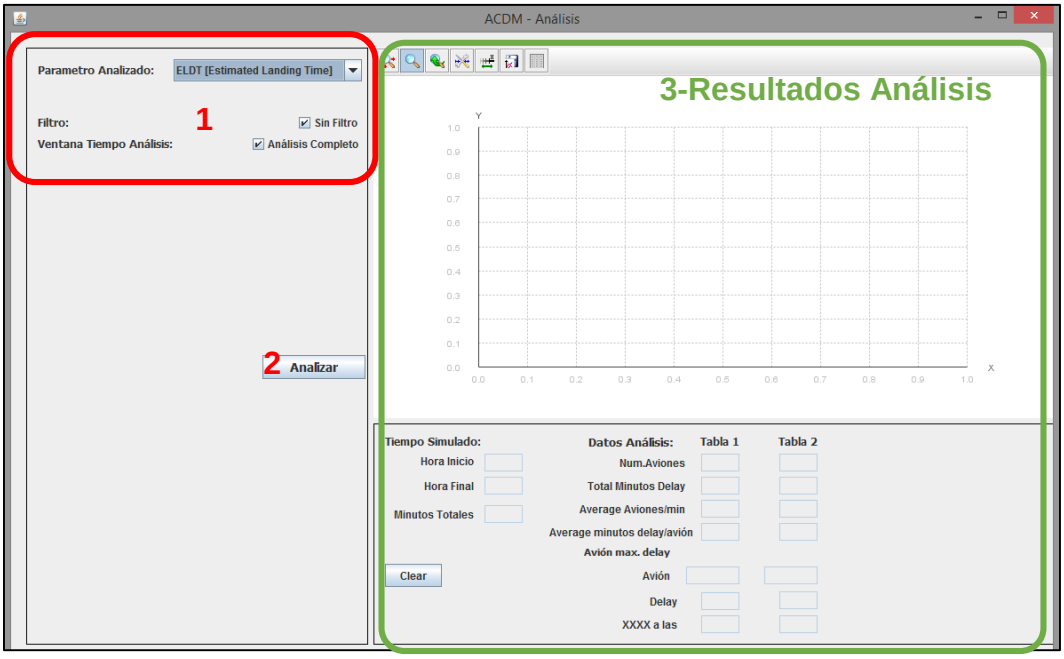


Figura D-16: Interfaz ACDM Análisis

- **1**: Parámetros Análisis: Aquí escogemos los parámetros del análisis que vamos a realizar. Tenemos de seleccionar que tiempo es el que queremos estudiar, si queremos aplicar algún filtrado, para estudiar solo alguna aerolínea o algún modelo en concreto. También podemos escoger una ventana de tiempo para realizar un estudio de unas horas en concretas. Seleccionaremos unos valores u otros según los resultados que queramos obtener.
- **2**: Analizar, al seleccionar esta opción el programa realizará un análisis de las tablas de resultados según los parámetros de análisis definidos (1).
- **3-Resultados Análisis:** Cada vez que el programa realice un estudio analizando los resultados de las tablas, mostrará por pantalla los valores encontrados. En la gráfica mostrará la cantidad de aviones afectados en el Parámetro analizado en un momento exacto de la simulación. Si pulsamos el botón Clear, el programa limpiará este panel para poder realizar un análisis de otros parámetros.

D.5 Parámetros que se pueden cambiar y no están en las interfaces.

Hasta el momento se han comentado los parámetros que se pueden cambiar desde documentos de entrada y desde las diferentes interfaces gráficas que encontramos en el programa.

Pero el programa utiliza una serie de variables internas que es importante tener en cuenta. Estás variables, aunque están dentro del código podrían ser modificadas por un usuario en caso de que este quisiera realizar los cálculos de la simulación con unos parámetros distintos a los que vienen predefinidos en el programa. A continuación expondremos una lista de estos parámetros, explicando la utilidad de cada uno y donde encontrarlo en caso de querer modificarlo.

D.5.1 Tiempos entre milestones

En la figura D-17 se muestran las comprobaciones que realiza el programa para asegurar que el tiempo entre los diferentes milestones respeta unos mínimos establecidos, tal y cómo se ha explicado durante la memoria. Podemos ver la relación que hay en el programa entre cada uno de los milestones de un avión.

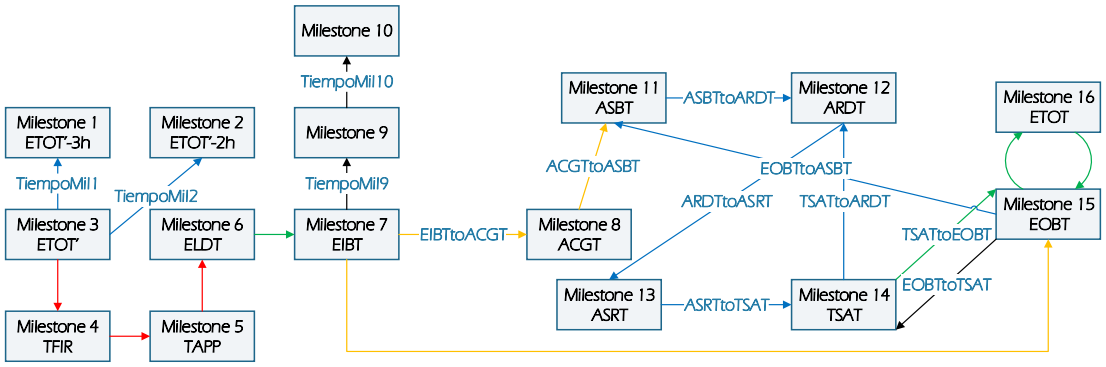


Figura D-17: Diagrama Relaciones Milestones

Estos parámetros se encuentran en agentes determinados, tienen un valor predeterminado dentro del código y, si se quieren modificar, es necesario cumplir una serie de restricciones para que el nuevo valor tenga coherencia con los otros tal y cómo se muestra en la tabla D-3.

Parámetro (Relación)	Agente	Valor (min)	Restricciones que debe cumplir
EIBTtoACGT	GH	3	< tiempoMil9
tiempoMil9	ATC	5	> EIBTtoACGT
tiempoMil10	ATC	3	-
ACGTtoASBT	GH	1	-
ASBTtoARDT	AO	20	-
ARDTtoASRT	AO	1	-
ASRTtoTSAT	AO	1	-
TSATtoARDT	AO	2	= ASRTtoTSAT + TSATtoEOBT
EOBTtoASBT	AO	27	= ASBTtoARDT + ARDTtoASRT + ASRTtoTSAT + TSATtoEOBT
TSATtoEOBT	Airport	5	= EOBTtoTSAT
EOBTtoTSAT	ATC	5	= TSATtoEOBT

Tabla D-2: Parámetros relaciones milestones

D.5.2 Retrasos aleatorios siguiendo una normal

Como se ha explicado en la memoria, el programa puede añadir un retraso aleatorio al finalizar cada uno de los milestones, suponiendo que por causas desconocidas el milestone sucede más tarde de lo previsto. Esto añade realismo a la simulación y hace que ya no sean simulaciones deterministas.

Si queremos aumentar la aleatoriedad o eliminar este retraso para hacer que el programa actúe de una manera determinista tendremos de entrar en el código para escoger los parámetros que deseamos.

Parámetros para decidir si queremos retrasos aleatorios:

- Variacionesnormales (GH): Decide si aplicar retrasos aleatorios en los milestones calculados por el GH.

- Variacionesnormales (AO): Decide si aplicar retrasos aleatorios en los milestones calculados por el AO.
- Variacionesnormales (ATC): Decide si aplicar retrasos aleatorios en los milestones calculados por el ATC.

Si ponemos todos estas variables en false, no tendremos retrasos aleatorios. Si decidimos tener alguno de estas 3 variables en true, el programa aplicará retrasos aleatorios. También podemos decidir el valor de σ^2 dentro del código en las variables de la tabla D-4:

Agente	***Variable a la que se aplicara la σ^2***	***Tiempo que sufre el retraso***
GH	VariacionNormalACGT	ACGT
AO	VariacionNormalARDT	ARDT
	VariacionNormalASRT	ASRT
	VariacionNormalTSAT	TSAT
ATC	VariacionNormalVuelo	TFIR
	VariacionNormalTFIR	TAPP
	VariacionNormalTAPP	ELDT
	VariacionNormalTaxiIn	EIBT
	VariacionNormalACGT	ACGT
	VariacionNormalEOBT	EOBT
	VariacionNormalTaxiOut	ETOT

Tabla D-3: Variables Variación Normal

D.5.3 Políticas PreDeparture Secuence

Tal y cómo se ha comentado a lo largo de la memoria el programa está preparado para usar dos políticas para realizar la Pre-departure Sequence.

La política de congestión a utilizar se escoge mediante el código del programa, de manera que si queremos preparar al programa para escoger una u otra tendremos de entrar en el código del agente ATC y escoger allí el valor de la variable *politicaCongestio*.

- *politicaCongestio* = 1: El primero que entra en la cola de salidas es el primero en salir, por lo que cuando un avión mueve su salida, tendrá de buscar un slot vacío en donde planificará su salida. Un solo avión sufre todo el retraso
- *politicaCongestio* = 2: El primer avión en salir será aquel que tenga intención de salir antes, moviendo los slots de los aviones posteriores. Los aviones se reparten los retrasos

D.6 Errores Comunes

A lo largo de las simulaciones es muy probable que el usuario se enfrente a diferentes situaciones, en las que no conseguirá obtener los resultados de la simulación que está solicitando. Los motivos a estos resultados negativos pueden ser muy variados, desde problemas con los datos de entrada, con los agentes o incluso por una mala elección de la velocidad de la simulación.

A continuación explicaremos una serie de errores comunes con los que se encuentran los usuarios del simulador, con el objetivo de poder ayudar a cualquier futuro usuario a evitarlos o a solucionarlos fácilmente.

- Usar un documento de retrasos erróneo: El programa permite aplicar retrasos leídos de un documento externo con el objetivo de poder repetir simulaciones. Pero puede pasar que usemos un documento de retrasos cuando estamos simulando un documento DDR2 distinto al que utilizamos en la primera simulación, por lo que es posible que estemos intentando aplicar retrasos en aviones que no existen en esta simulación.

- Problemas con el documento DDR2: El documento DDR2 que leemos del documento extraído de la página de EUROCONTROL es un documento muy extenso, y a veces resulta que este viene con errores de escritura desde EUROCONTROL. Estos errores pueden causar problemas y provocar que el programa sea incapaz de leer el documento. Por lo que si nos aparecen problemas al leer este documento debemos considerar la opción de que el error venga directamente del documento escrito por EUROCONTROL.

  Es probable que este error se produzca en un número muy limitado de aviones, pero hace que toda la lectura del archivo se vea comprometida. Para comprobar si ese es el problema basta con intentar leer el mismo documento pero para un aeropuerto diferente, si haciendo pruebas con varios aeropuertos el documento se lee bien en alguno de ellos, quiere decir que debemos volver a descargarlo de DDR2 para poder hacer una simulación del tráfico aéreo al aeropuerto que queríamos en primer lugar

- Error con las aerolíneas y las empresas de ground handling con agente propio en las *Run Configurations*: Tal y cómo se ha explicado en el apartado de *Como ejecutar el programa*, podemos crear aerolíneas y empresas de ground handling con agente propio. Estos agentes se tienen que indicar tanto en las *Run Configurations* cómo en el documento Airport.txt, habrá errores en las simulaciones cuando tenemos un AO o GH solo en uno de los dos apartados.

  Si creamos un agente en las *Run Configurations*, pero este no aparece en el documento Airport.txt, el programa creará el agente, pero nadie se comunicará con él. Por otra parte, si añadimos una aerolínea o una empresa de ground handling en el documento Airport.txt, pero está no

aparece en las *Run Configurations*, el programa no creará el agente de dicha empresa o aerolínea, pero sí que intentará mandarle mensajes, causando errores en la simulación.

- Error con el tiempo de simulación: Al empezar con la simulación el usuario puede escoger la velocidad de simulación. Al escoger este parámetro estamos decidiendo el tiempo que el simulador dejara entre milestone y milestone.
  Para simulaciones con pocos aviones, este parámetro tiene poca importancia, pero en simulaciones más grandes, la cantidad de cálculos que realizará el simulador entre milestone y milestone es muy grande, pues tendrá que tener en cuenta más aviones.

  Por este motivo, cuantos más aviones simulamos, mayor debería de ser el tiempo entre milestones (velocidad de simulación). Si este tiempo fuera muy pequeño, los agentes acumularían más mensajes de los que pueden ejecutar, y en algún momento la lista de mensajes acumulados superaría la capacidad de estos agentes a guardar mensajes y el programa se pararía. Por esto es aconsejable ir aumentando este tiempo en función de la cantidad de aviones que se simularán. Por ejemplo, para simulaciones pequeñas podremos escoger 300ms entre milestones, si la simulación envuelve a 600 aviones, no es aconsejable bajar el tiempo entre milestones por debajo de los 1500ms.

  Este parámetro, al afectar a la velocidad de procesamiento del programa, también dependerá del PC en el que se ejecute. En un PC con una capacidad de procesamiento mayor podemos disminuir el tiempo entre milestones sin problemas.

D.7 Ejemplo Simulación paso a paso

Este apartado no sería estrictamente necesario para comprender cómo se realiza una simulación, puesto que consideramos que con los apartados anteriores se podría ejecutar el programa perfectamente. Pero en caso de que el usuario se vea perdido, aquí tenemos un ejemplo de cómo se realizaría una simulación paso a paso, cómo manejar las interfaces y los parámetros para llegar a unos resultados y poder analizarlos de manera correcta.

Una vez tenemos las Run configurations preparadas, y los documentos de entrada escritos correctamente ya podemos iniciar la simulación. A continuación encontraremos explicaremos paso a paso cómo podemos realizar una simulación. Explicaremos todo lo que encontramos a lo largo de la simulación, y todas las decisiones que puede tomar un usuario durante una simulación.

Al iniciar una simulación siempre se nos abrirá la interfaz del simulator input, figura D-18:

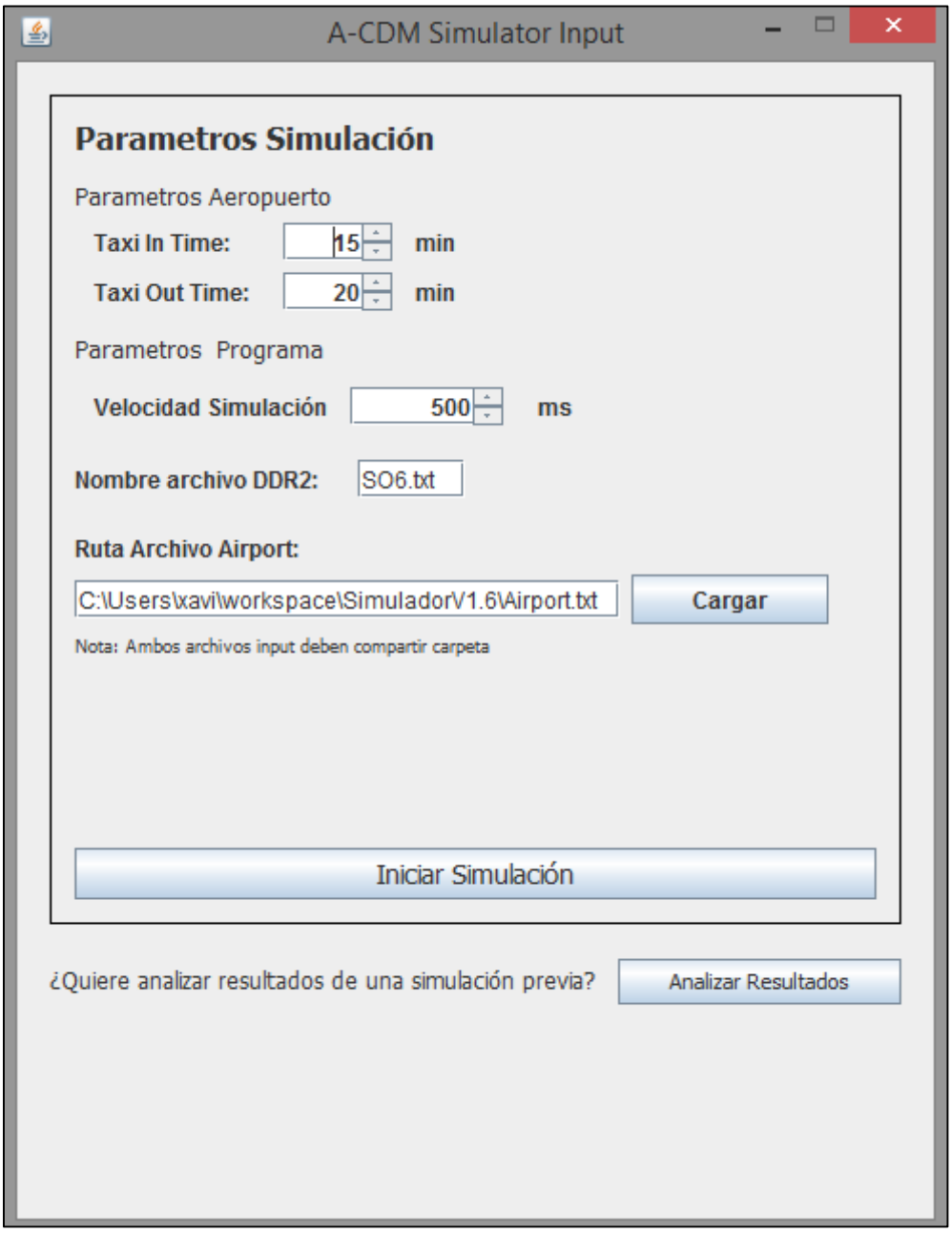


Figura D-18: Interfaz Simulator Input

En la interfaz D-18 a simple vista podemos observar 3 botones y una serie de textBox y spinners. El botón *Iniciar Simulación* no se puede pulsar hasta que no se carguen los datos.

Lo primero que nos tenemos que preguntar al empezar es si queremos realizar una nueva simulación o solo queremos estudiar los resultados de una simulación ya realizada.

D.7.1 Estudio nueva simulación

Si vamos a realizar una nueva simulación la primera tarea será responder a las preguntas de los textBox y los spinners de la interfaz, para facilitar al programa los parámetros de Simulación deseados.

Primero deberemos completar los parámetros del Aeropuerto, añadiendo el tiempo de Taxi, el programa diferenciará entre el tiempo de entrada (Taxi in Time) y el de salida (Taxi Out Time), ambos valores serán números enteros e indican la cantidad de minutos que se tarda en realizar el taxi.

A continuación completaremos los parámetros del programa, únicamente se trata de un spinner que indica la velocidad a la que saltará el timer del MilestoneTrigger, este valor será el que marcará el tiempo total de la simulación. Si es demasiado pequeño puede provocar que el programa se sature en grandes simulaciones. Siempre será un valor entero que indicará los milisegundos entre cada lanzamiento de milestone. Se aconseja que este valor este alrededor de los 250ms para simulaciones pequeñas (100 aviones o menos) y que se escojan valores mayores llegando a 2000ms para simulaciones grandes (900 aviones).

Por último rellenaremos los textBox, figura D-19, para que el programa pueda encontrar los archivos necesarios.

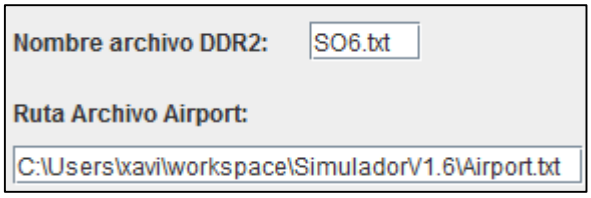


Figura D-19: TextBox Entrada archivos Input

Deberemos poner el nombre del documento extraído de Eurocontrol, que tiene un formato SO6 (tener en cuenta si lo tenemos en versión .txt o versión .so6), que queramos simular, y la ruta completa del documento Airport que leerá el programa.
Esta ruta es la que se usará también para leer los otros documentos y donde se escribirán todos los documentos de salida.

Una vez los parámetros estén preparados tendremos de decir-le al programa que los cargue (Botón *Cargar*).

Si el nombre del archivo DDR2 es incorrecto o este no se encuentra en la misma carpeta que el archivo Airport nos aparecerá un messageBox cómo el de la figura D-20.

Figura D-20: Mensaje Error Ruta DDR2

Si la ruta del documento Airport.txt es incorrecta aparecerá la figura D-21 en pantalla:

Figura D-21: Mensaje Error Ruta Airport

Cuando el programa pueda encontrar el archivo Aeropuerto.txt, podremos observar que la interfaz Simulator Input muestra por pantalla un label avisándonos de que se están procesando los datos del DDR2, y a la vez ha bloqueado los parámetros introducidos, ya que serán los que se utilizarán en la simulación.

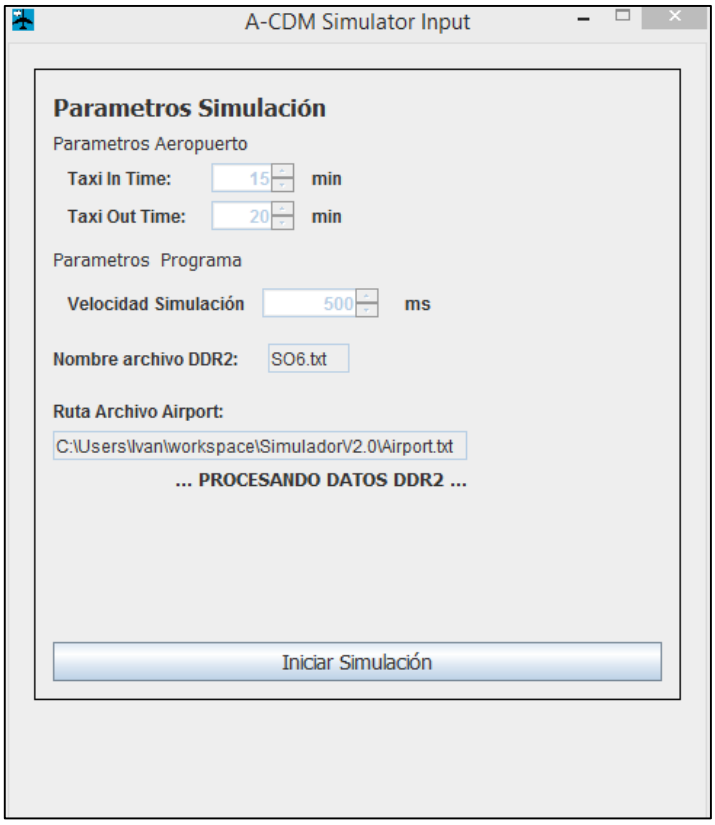


Figura D-22: Interfaz procesando DDR2

Al terminar de procesar los datos DDR2 el programa nos avisará y nos mostrará 2 messageBox.

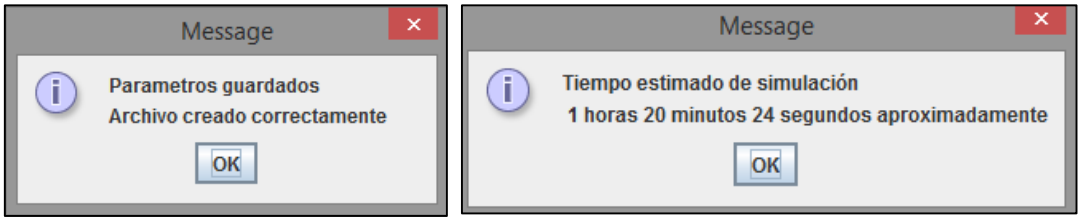


Figura D-23: Mensajes Aviso Parámetros Simulación

El primero avisa que se han creado correctamente los documentos intermedios, y el segundo informa de cuál será el tiempo que requerirá el simulador para realizar la simulación completa.

Ahora la Interfaz Simulator Input se ha actualizado a la figura D-24:

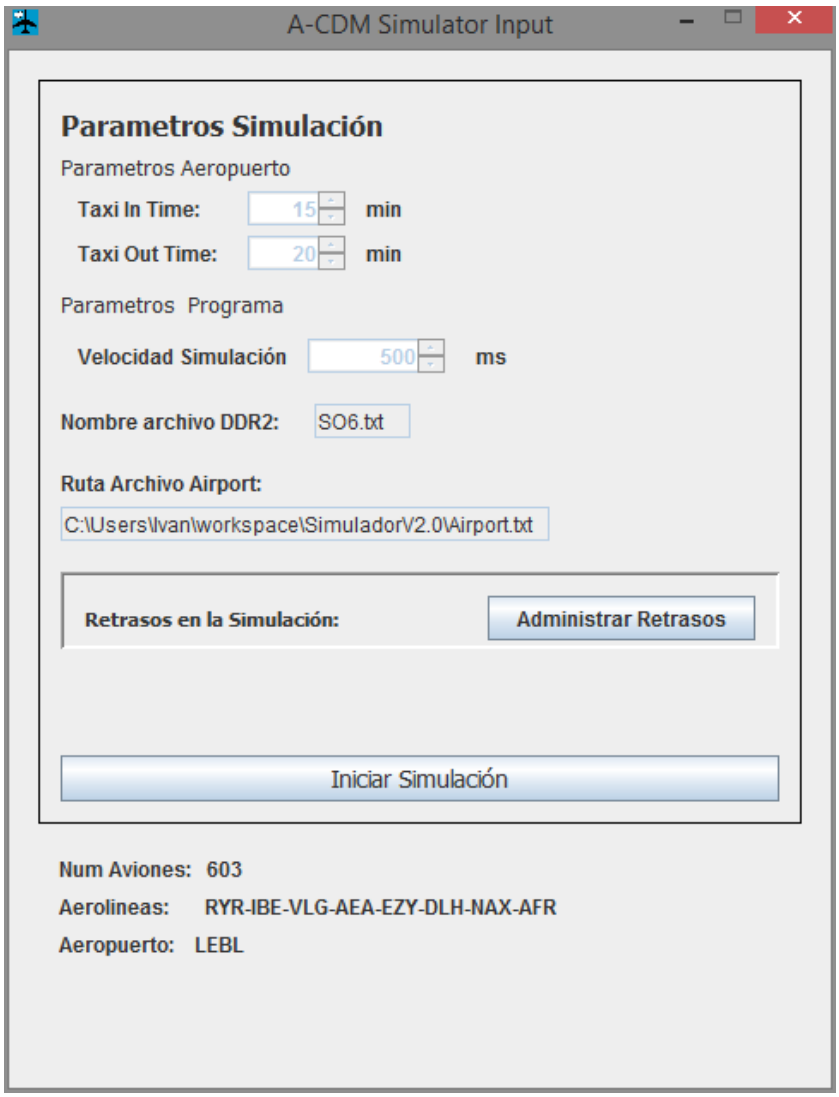


Figura D-24: Interfaz Sim.Input Actualizada

Al tener el documento DDR2 leído, podremos observar que en la interfaz Simulator Input, ya no podemos seleccionar la opción *Analizar Resultados*, pues se considera que ya se ha decidido que se realizará una simulación completa.
Ahora, una vez leído el documento podemos observar que la interfaz nos facilita información sobre los documentos leídos cómo el aeropuerto que estudiamos, el número de aviones que se simularán y el nombre de las aerolíneas con agente propio.

También podemos observar en la figura D-24, que aparece la opción de *Administrar Retrasos*. En este punto tendremos de decidir si queremos que el simulador tenga en cuenta algún retraso externo escogido por el usuario.

D.7.1.1 Simulación con retrasos

Si decidimos añadir algún retraso pulsaremos el Botón *Administrar Retrasos.* En este momento se abrirá una nueva interfaz, *Retrasos Simulación* (figura D-25) desde donde podremos trabajar con los retrasos que se aplicarán durante la simulación.

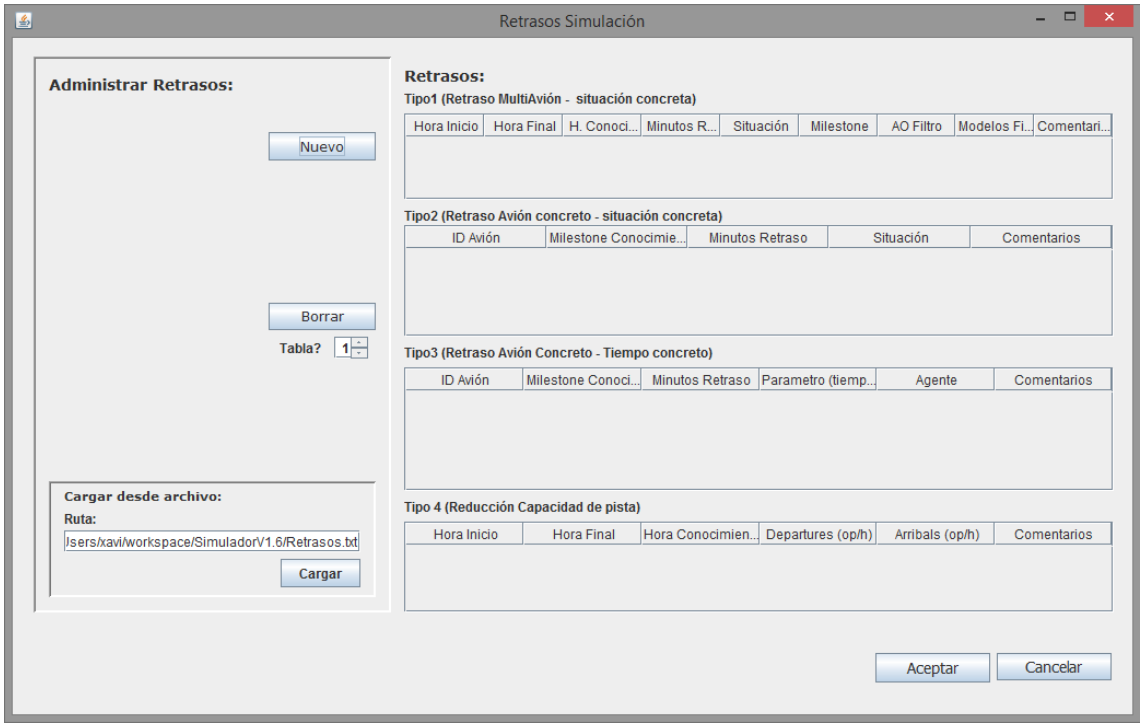


Figura D-25: Interfaz Administrar Retrasos

En esta interfaz podemos observar 2 zonas:

- Retrasos: Está situada a la derecha, y muestra 4 tablas con los retrasos que se aplicarán en este momento. Tenemos puestas 4 tablas, pues cada una muestra solo un tipo de retrasos (Mirar apartado sobre Retrasos en la memoria):

- Tabla 1: Retrasos Tipo 1 (Retraso MultiAvión – situación concreta)
- Tabla 2: Retrasos Tipo 2 (Retraso Avión concreto – situación concreta)
- Tabla 3: Retrasos Tipo 3 (Retraso Avión concreto – tiempo concreto)
- Tabla 4: Retrasos Tipo 4 (Reducción Capacidad de pista)

- Administrar Retrasos: Está situado a la izquierda. Desde aquí podemos trabajar sobre los retrasos que aparecen en la parte izquierda de la interfaz. Podemos realizar 3 acciones distintas que modificarán en alguna manera los retrasos que aparecen en las tablas, que son los que se aplicarán en el simulador.

Cargar Retrasos de un documento:

Si queremos cargar retrasos de un documento externo, escrito por nosotros o escrito por el programa en otra simulación, nos centraremos en el apartado de *Cargar desde archivo.*

Tenemos un textBox donde pondremos la ruta completa para que el programa encuentre el documento con la lista de retrasos. Si estos están escritos correctamente siguiendo las reglas descritas en el apartado sobre Retrasos en la memoria podremos ver cómo en las tablas aparecen estos retrasos cargados.

En caso de que la ruta sea incorrecta aparecerá un messageBox cómo el de la figura D-26:

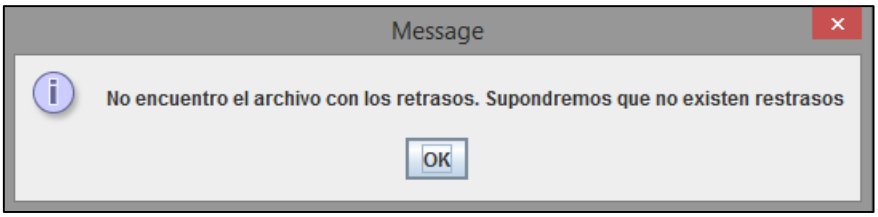


Figura D-26: Mensaje Error Ruta Retrasos

Solo se pueden cargar retrasos de un documento exterior una vez por simulación

Crear un retraso nuevo

Si queremos crear un retraso nuevo pulsaremos el botón *Nuevo* de la interfaz *Retrasos Simulacion*. Al crear un nuevo retraso se abrirá una nueva interfaz, *Interfaz Definir Retraso,* figura D-27, desde donde podremos escoger todos los parámetros de este nuevo retraso.

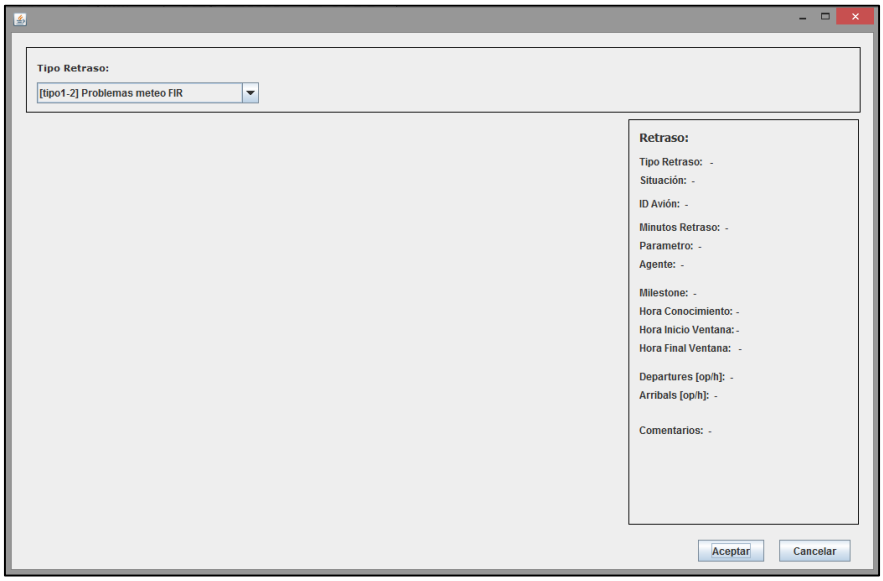


Figura D-27: Interfaz Crear Retraso

En esta nueva interfaz observamos un ComboBox para escoger el tipo de retraso, dos botones, uno de aceptar y otro de cancelar, y una caja con información sobre los Retrasos a la izquierda.

La primera acción que necesaria en esta interfaz es escoger el tipo de retraso que aplicaremos en el ComboBox que nos dará una serie de opciones tal y cómo vemos en la figura D-28

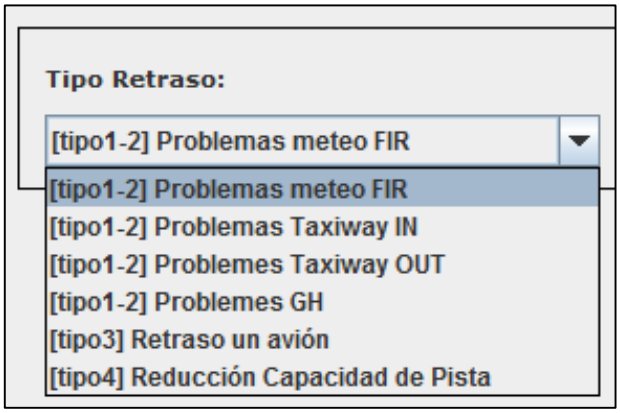


Figura D-28: Lista elección tipo de Retrasos

Retrasos tipo 1 y tipo 2

Como podemos observar, tenemos 4 opciones en la que el retraso que se aplicará será de tipo 1 o tipo 2, al escoger una opción u otra, estamos decidiendo cual será el valor del parámetro *Situación* en el retraso. Por lo que ya estamos decidiendo cual será el milestone afectado y que agente aplicará el retraso.

Al seleccionar cualquiera de estas opciones en el ComboBox, la interfaz mostrará un nuevo apartado, figura D-29, donde escoger los parámetros del retraso:

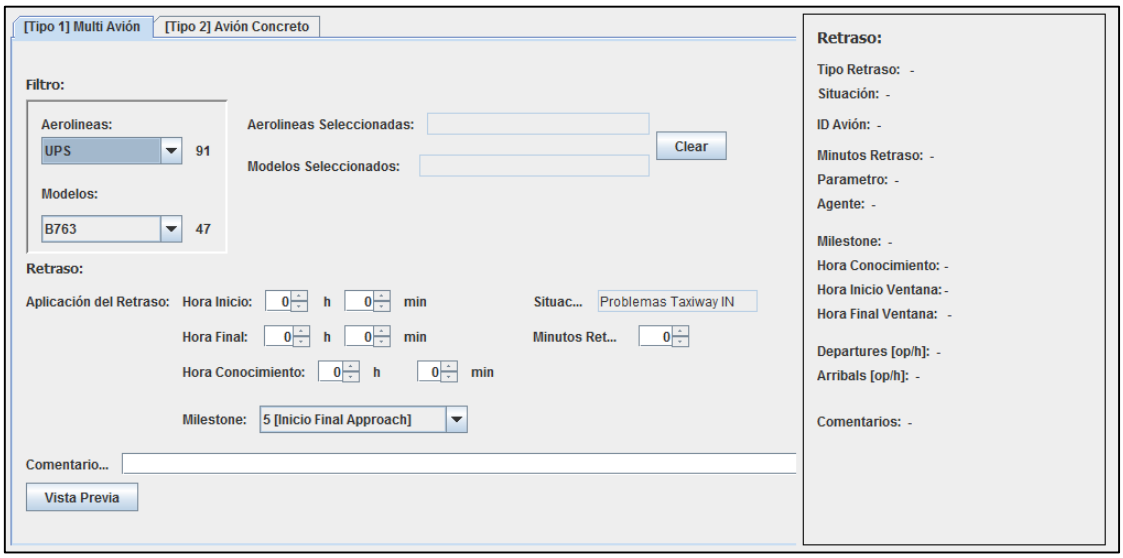


Figura D-29: Interfaz definir retrasos tipo 1 o tipo 2

Antes de empezar a rellenar las propiedades del retraso, tenemos de escoger si queremos trabajar en los retrasos de tipo 1, que afectará a más de un avión (figura D-30); o de tipo 2 (figura D-32) donde tendremos un apartado donde podremos buscar el avión que decidimos retrasar (donde podremos filtrar por aerolínea y modelo para buscarlo).

Retrasos tipo 1

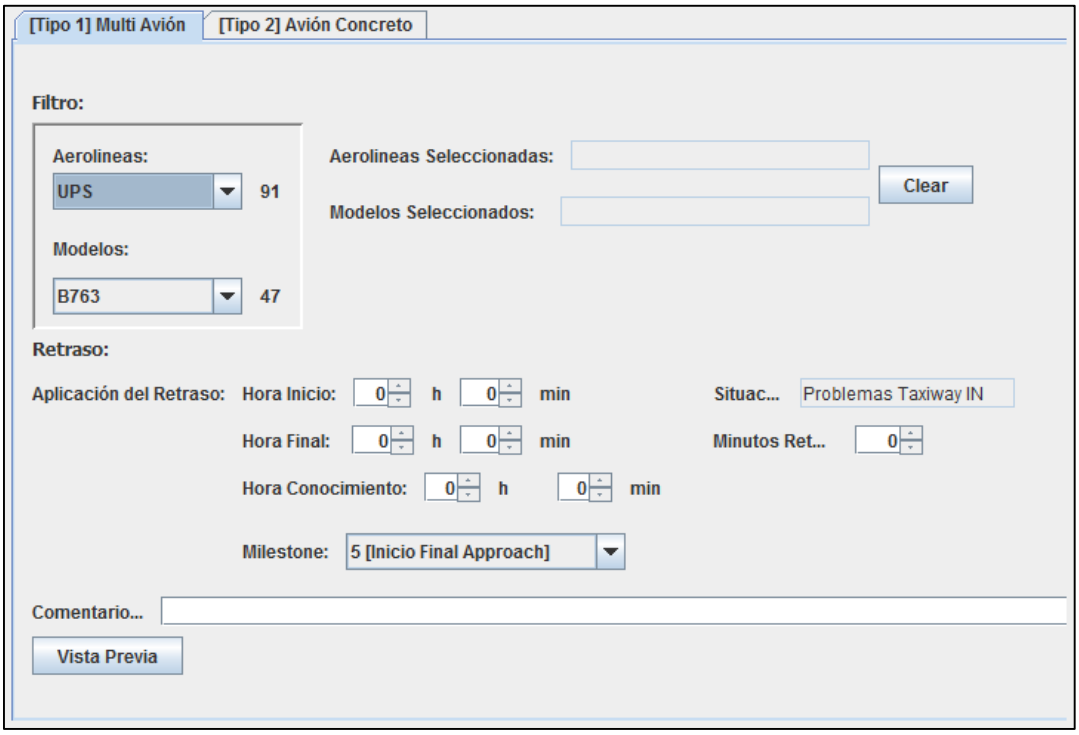


Figura D-30: Panel Parámetros Retraso Tipo 1

Hay que tener en cuenta que el comboBox *Milestone* es para seleccionar el Milestone de conocimiento, el programa tiene una protección de coherencia por la cual no nos dejará seleccionar que el milestone de conocimiento sea posterior al afectado.

De la misma manera, el programa no dejará al usuario no ser coherente con los tiempos de esta ventana. No dejará al usuario definir retrasos si no se cumple:

$$H.Conocimiento \leq H.Inicio < H.Final$$

Los filtros son para escoger los modelos y las aerolíneas que se ven afectados por el retraso. Si no escogemos ningún filtro estamos indicando que todos los aviones resultarán afectados, mientras que si escogemos filtros de modelo y aerolínea, estamos diciendo que solo los aviones que cumplan ambos requisitos a la vez se verán afectados. En caso de seleccionar un filtro no deseado podemos limpiar los filtros pulsando el botón *Clear*.

Una vez tengamos todos los parámetros escogidos pulsaremos a *Vista Previa*, en este momento veremos cómo la lista de parámetros se actualiza mostrando los parámetros que se han seleccionado, figura D-31.

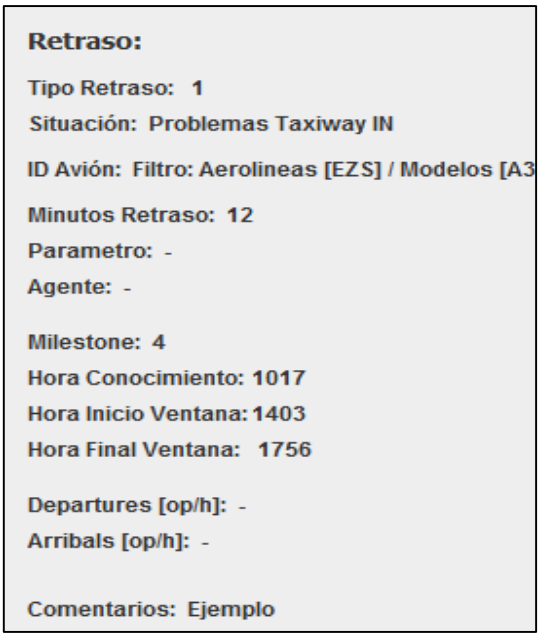


Figura D-31: Ejemplo Vista Previa Retraso Tipo1

Retrasos tipo 2

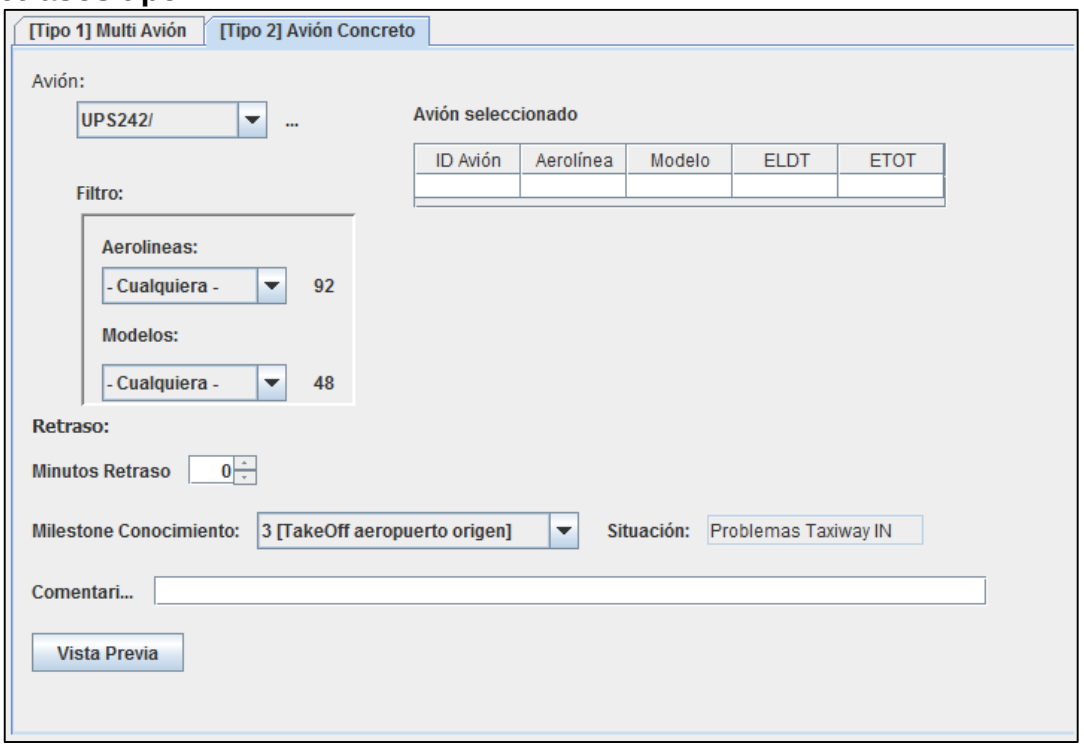


Figura D-32: Panel Parámetros Retraso Tipo2

Este panel de parámetros es muy similar al anterior, solo cambia el apartado de cómo seleccionar el/los aviones afectados por el retraso.

Podemos ver que mientras antes teníamos un filtro y definíamos una ventana de tiempos para seleccionar los aviones afectados, ahora únicamente tenemos un filtro de búsqueda.

Este filtro nos ayudará a seleccionar el avión a retrasar, pues la lista para seleccionar el avión dependerá de los parámetros elegidos en el filtro. Una vez

elegido, el avión y sus tiempos principales aparecen en la tabla en la parte superior del panel.

Una vez tengamos todos los parámetros escogidos pulsaremos a *Vista Previa*, en este momento veremos cómo la lista de parámetros se actualiza mostrando los parámetros que se han seleccionado, figura D-33:

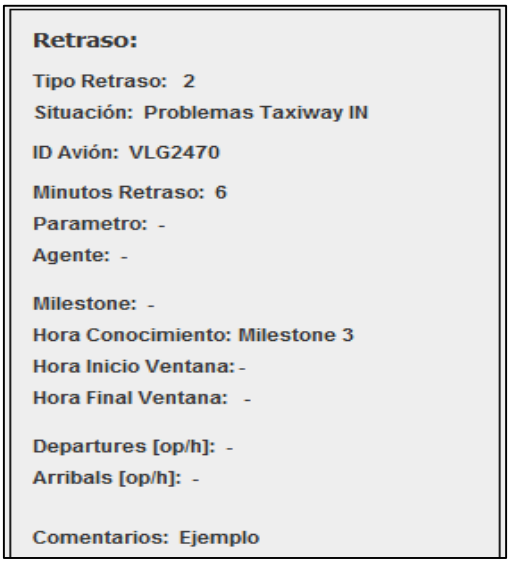


Figura D-33: Ejemplo Vista Previa Retraso Tipo2

Retrasos tipo 3

Si decidimos definir un retraso de tipo 3, definiremos los parámetros del retraso en la interfaz de la figura D-34

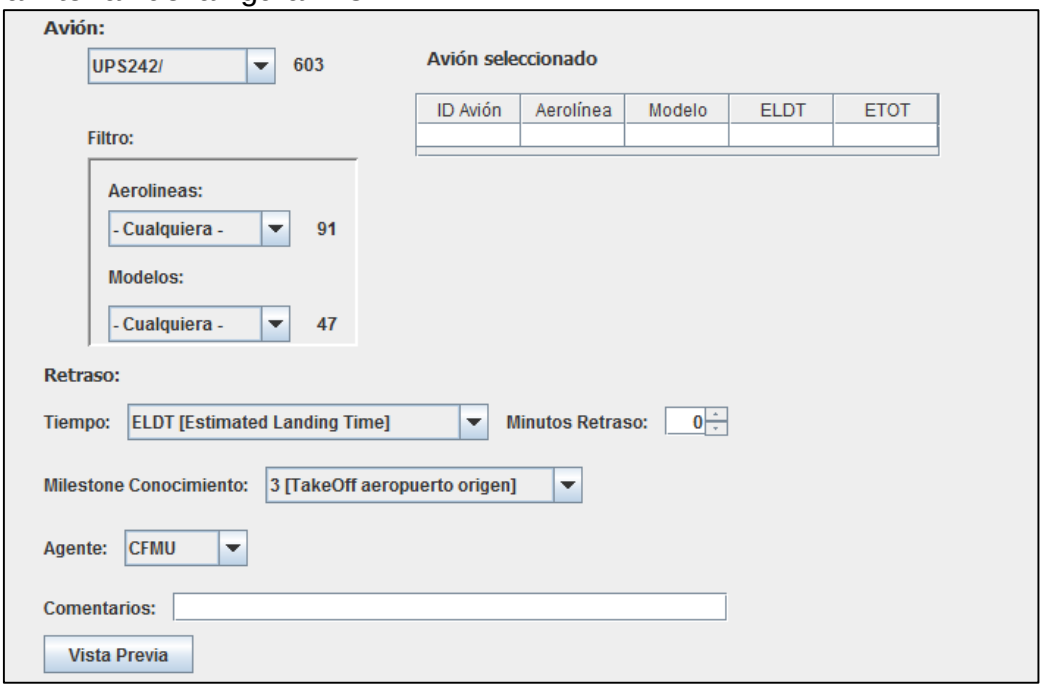


Figura D-34: Panel Parámetros Retraso Tipo3

En este panel primero seleccionaremos el avión afectado, i a continuación decidiremos el tiempo del avión en donde aplicaremos el retraso

A la hora de elegir el milestone de conocimiento, el programa solo nos dejará si somos coherentes con esta elección, no permitirá que el milestone sea posterior al tiempo afectado.

En el ComboBox *Agente* seleccionamos el agente encargado de aplicar el retraso.

Una vez tengamos todos los parámetros escogidos pulsaremos a *Vista Previa*, en este momento veremos cómo la lista de parámetros se actualiza mostrando los parámetros que se han seleccionado. Un ejemplo de lo que veríamos con un retraso tipo 3 se encuentra en la figura D-35:

Figura D-35: Ejemplo Vista Previa Retraso Tipo3

Retrasos tipo 4

Si decidimos definir un retraso de tipo 4, definiremos los parámetros del retraso en la interfaz de la figura D-36

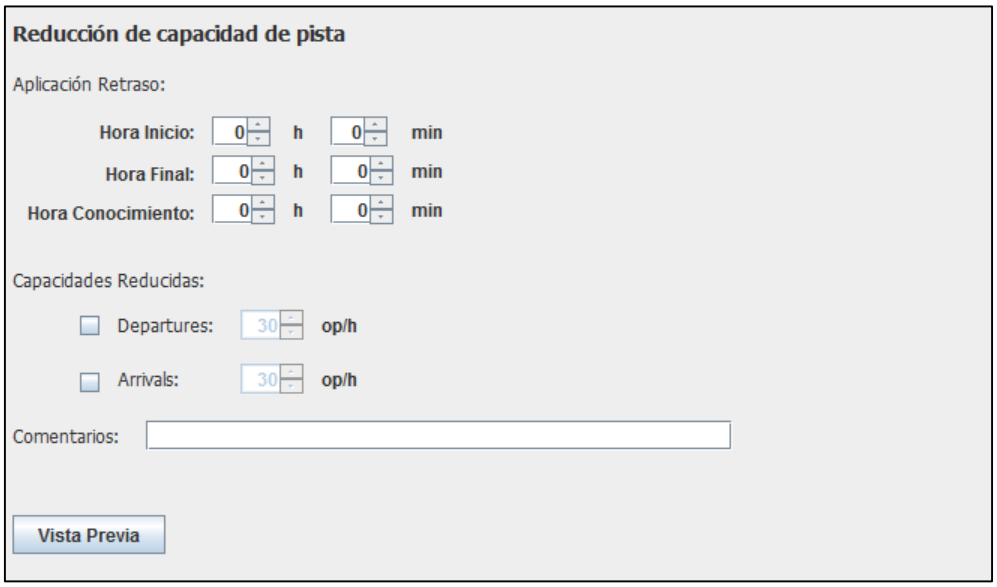


Figura D-36: Panel Parámetros Retraso Tipo4

Una vez tenemos seleccionado el retraso tipo 4 empezamos a rellenar todos los apartados.

De la misma manera que teníamos en el panel de parámetros de los retrasos tipo 1, el programa solo permitirá definir la ventana de tiempo si esta es coherente con sus tiempos.

$$H.Conocimiento \leq H.Inicio < H.Final$$

En Capacidades Reducidas escogemos el nuevo valor de la capacidad de pista, al tratarse solo de reducciones solo se puede reducir el nuevo valor de operaciones/hora. Como se han definido 30 op/h cómo la nominal y se pretende definir una reducción de la capacidad de la pista, la interfaz no permitirá introducir números mayores a 30.

Una vez tengamos todos los parámetros escogidos pulsaremos a *Vista Previa*, en este momento veremos cómo la lista de parámetros se actualiza mostrando los parámetros que se han seleccionado, figura D-37

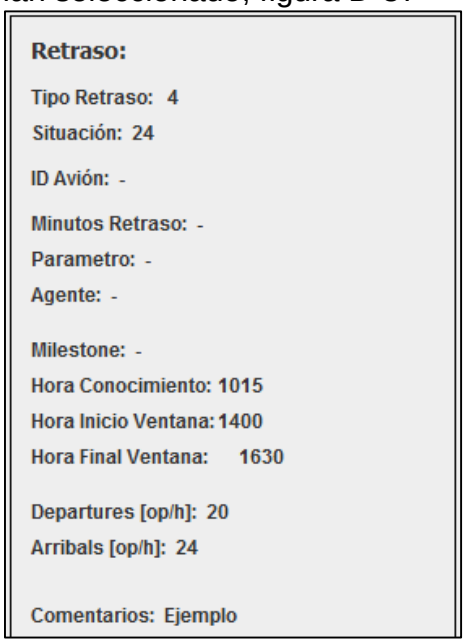


Figura D-37: Ejemplo Vista Previa Retraso Tipo4

Aceptar retraso

Una vez hemos definido los retrasos tenemos de pulsar el botón *aceptar* de la interfaz *Definir Retraso.*
Este botón solo funciona si se han definido previamente los parámetros y estos aparecen en la lista de parámetros de la interfaz. Una vez aceptado el retraso volveremos a la interfaz *Retrasos Simulación.*

Si no se cumplen estas condiciones el programa nos mostrara un messageBox cómo el de la figura D-38

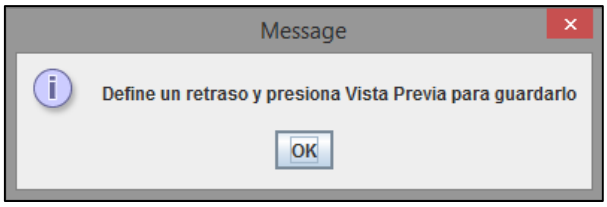


Figura D-38: Mensaje Error Definición Retrasos

Cancelar retraso

Si cambiamos de idea y decidimos no añadir el retraso que estábamos definiendo podemos pulsar en cualquier momento el botón *cancelar* de la interfaz *Definir Retraso*, para salir de la interfaz y volver de nuevo a la interfaz *Retrasos Simulación*.

Borrar Retraso

En la interfaz *Retrasos Simulación* también tenemos la opción de borrar un retraso ya añadido. Para borrar-lo es necesario seleccionar la tabla de la que queremos borrar el retraso en el spinner "*¿Tabla?"* a la vez que seleccionamos el valor que queremos borrar.

Al pulsar el botón *Borrar* eliminaremos el retraso seleccionado de tipo 3. Pero si por lo que fuera intentásemos borrar un elemento sin seleccionarlo o sin indicar la tabla correcta en el spinner *"¿Tabla?"* el programa nos avisaría mostrándonos el messageBox de la figura D-39

Figura D-39: Mensaje Error Borrar Retraso

Cancelar Retrasos

Si cambiamos de idea y decidimos no aplicar los retrasos que estábamos definiendo podemos pulsar botón *cancelar* de la interfaz *Retrasos Simulación*. Una vez pulsado el botón volveremos a la interfaz *Simulator Input* y el programa nos avisará de que esos retrasos no serán definidos y utilizados en la simulación.

Aceptar Retrasos

Una vez las tablas de la interfaz *Retrasos Simulación* son los que queremos aplicar pulsaremos el botón *aceptar* que encontramos en esta misma interfaz.

Solo se aplicarán los retrasos que aparezcan en las tablas en el momento de pulsar el botón. Una vez pulsado el botón, el programa reescribirá el documento retrasos.txt con los nuevos retrasos definidos, y nos mostrará una interfaz nueva, donde nos preguntará si queremos guardar el documento. En este punto nos da la opción de guardar los retrasos en una nueva dirección tal y cómo vemos en la figura D-40.

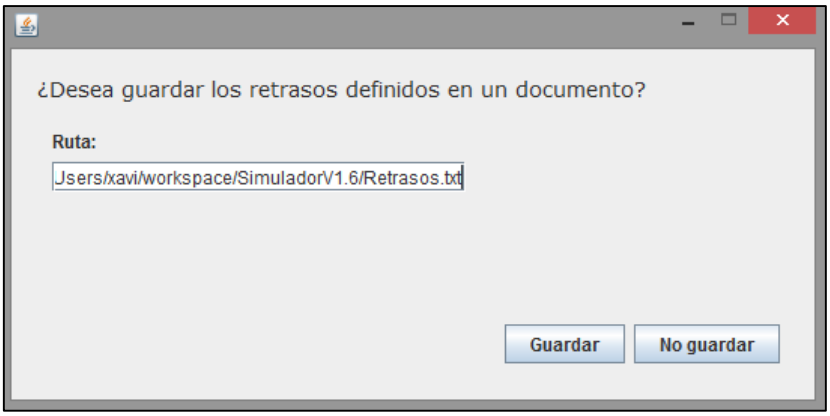


Figura D-40: Interfaz Guardar Retrasos

Si queremos otra copia de los retrasos definidos únicamente tenemos de escribir la ruta en el textBox, y pulsar el botón *Guardar,* si no queremos una copia de seguridad de los retrasos pulsaremos el botón *No Guardar*. En cualquiera de las dos situaciones el programa nos llevará a la interfaz Simulator Input de nuevo.

Si volvemos a entrar a administrar retrasos podemos cambiar los retrasos definidos previamente o cancelar los retrasos si finalmente decidimos no aplicarlos.

D.7.1.2 Iniciar Simulación

Tengamos o no retrasos el siguiente paso es iniciar la simulación y dejar que el programa corra realizando la simulación de todos los aviones leídos en el documento de Eurocontrol. Para iniciar la simulación pulsaremos el botón *Iniciar Simulación* en la interfaz *ACDM-SimulatorInput.*

Una vez iniciada la simulación nos aparecerá una tabla en la pantalla donde veremos el estado de todos los aviones que están en el sistema. Figura D-41.

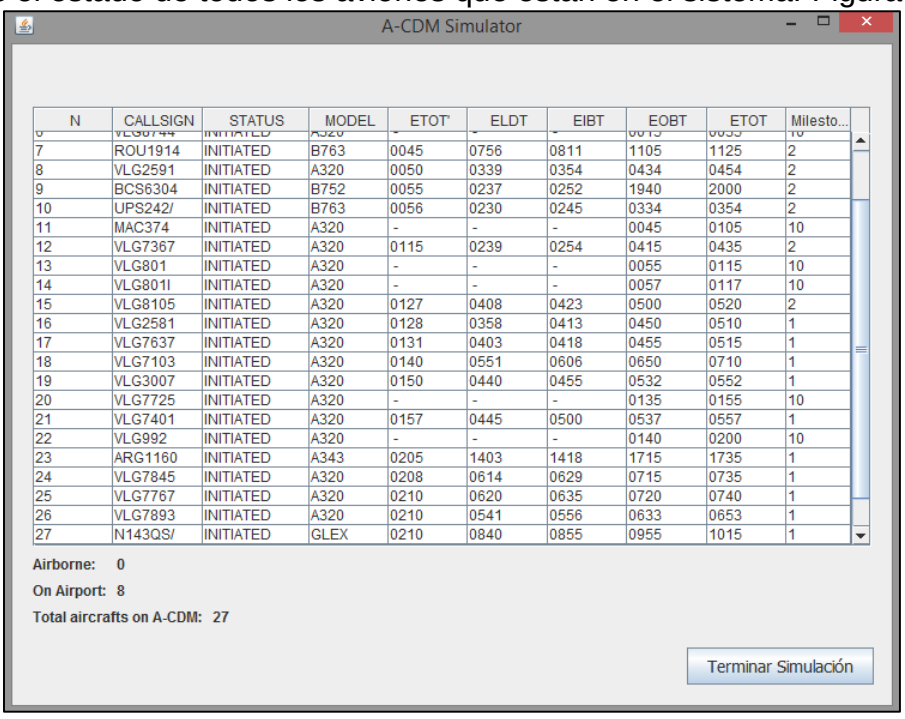


Figura D-41: Interfaz Simulación

En esta interfaz, vemos el estado de los aviones a tiempo real. También tenemos un pequeño panel informativo abajo a la izquierda, que nos indica el número de aviones que tenemos dentro del aeropuerto en un momento dado, y el número de aviones que ya han despegado de nuestro aeropuerto. También muestra un contador de todos los aviones que han entrado en el sistema por el momento.

Está interfaz es solo de transición mientras el simulador realiza la simulación completa. Pero también nos ofrece la opción de parar el programa a mitad de una simulación.

Realizar simulación completa

En principio en esta interfaz no realizaremos ninguna acción, su función es mantenernos informados de cómo avanza la simulación. En cuanto el programa termine sus cálculos instantáneamente se nos abrirá una nueva interfaz con la tabla de resultados, interfaz *A-CDM resultados.*

Terminar Simulación

Como se ha comentado, mientras el programa realiza la simulación, nos da la opción de parar los cálculos en cualquier momento pulsando el botón *Terminar Simulación* en la interfaz *A-CDM simulator*.

Al pulsar este botón el programa abrirá una nueva interfaz con la tabla de resultados, interfaz *A-CDM resultados.* Esta tabla solo mostrará los resultados calculados hasta el momento de parar la simulación, que es lo que se ha simulado.

D.7.2 Leer resultados de un documento externo.

Si nuestro objetivo es visualizar unos resultados ya calculados en otra simulación, pulsaremos el botón *Analizar Resultados* en la interfaz *Simulator Input,* entonces, la interfaz mostrará un nuevo apartado, figura D-42, donde nos da la opción de escoger la ruta de donde cargar el documento de resultados de una simulación ya realizada.

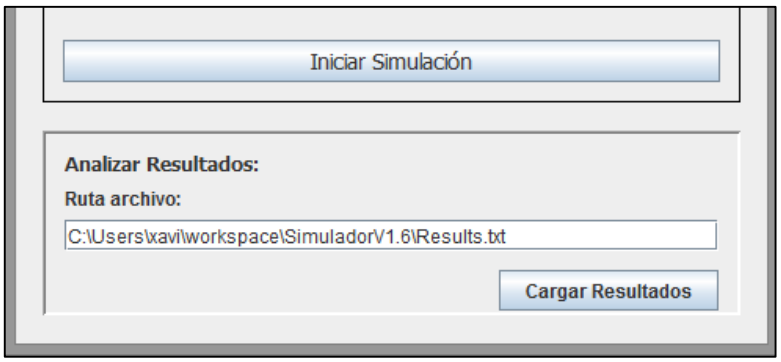


Figura D-42: Captura Cargar Resultados Externos

Una vez tengamos escrita la ruta donde encontrar el documento de retrasos, pulsaremos el botón *Cargar Resultados.* En este momento el programa abrirá la interfaz *A-CDM Results* y nos mostrará una tabla con los resultados leídos en el documento.

D.7.3 Mostrar Resultados

A la interfaz *A-CDM Results,* es posible acceder de 3 maneras distintas

(1) Realizando una simulación completa.

(2) Realizando un fragmento de una simulación

(3) Leyendo un documento de una simulación ya realizada

Sea cual sea nuestra situación la interfaz que se abre será la de la D-43

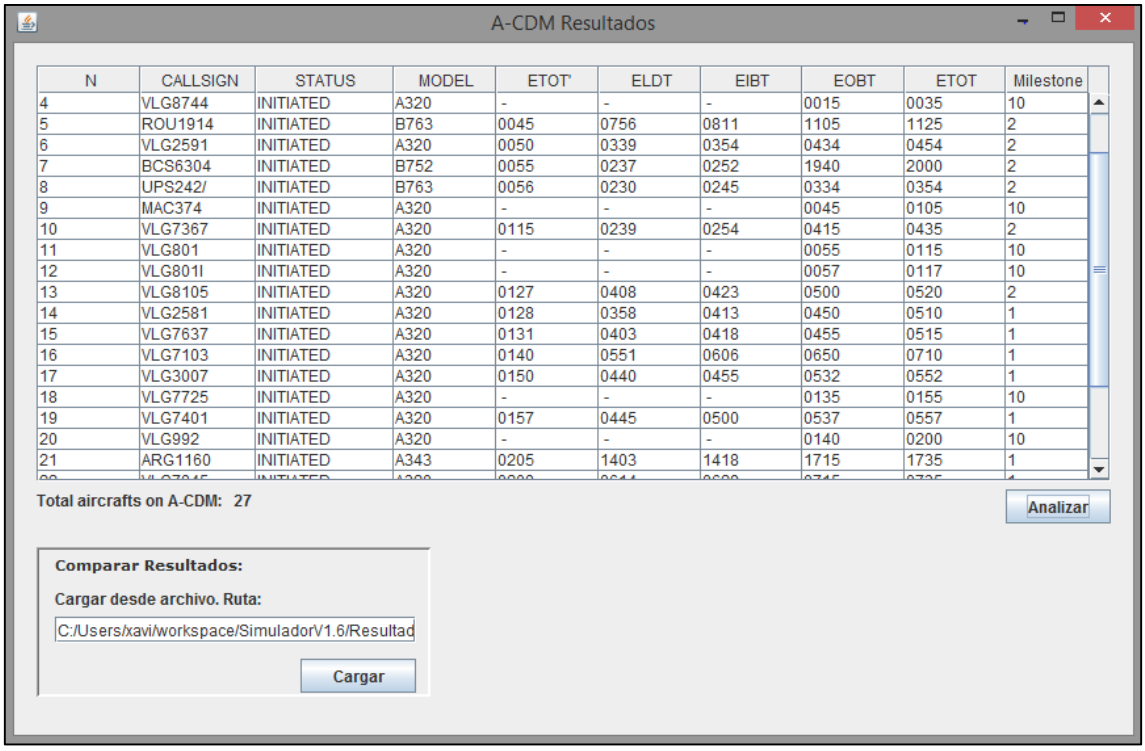


Figura D-43: Interfaz Resultados

En la parte superior vemos la tabla con los resultados de la simulación realizada o los valores leídos en el documento.

Esta interfaz también nos da la opción de cargar otro documento de resultados para comparar ambos. En caso de querer comparar resultados, debemos añadir la ruta del archivo de resultados y darle al botón de Cargar

En este momento el programa leerá el documento que se encuentre en la ruta escrita en el textBox, y nos mostrará por pantalla las dos tablas a la vez. A la izquierda la tabla de resultados que ya teníamos anteriormente, y a la derecha la nueva, para así permitir al usuario que compare los datos de ambos tráficos aéreos simulados, tal y cómo se ve en la figura D-44:

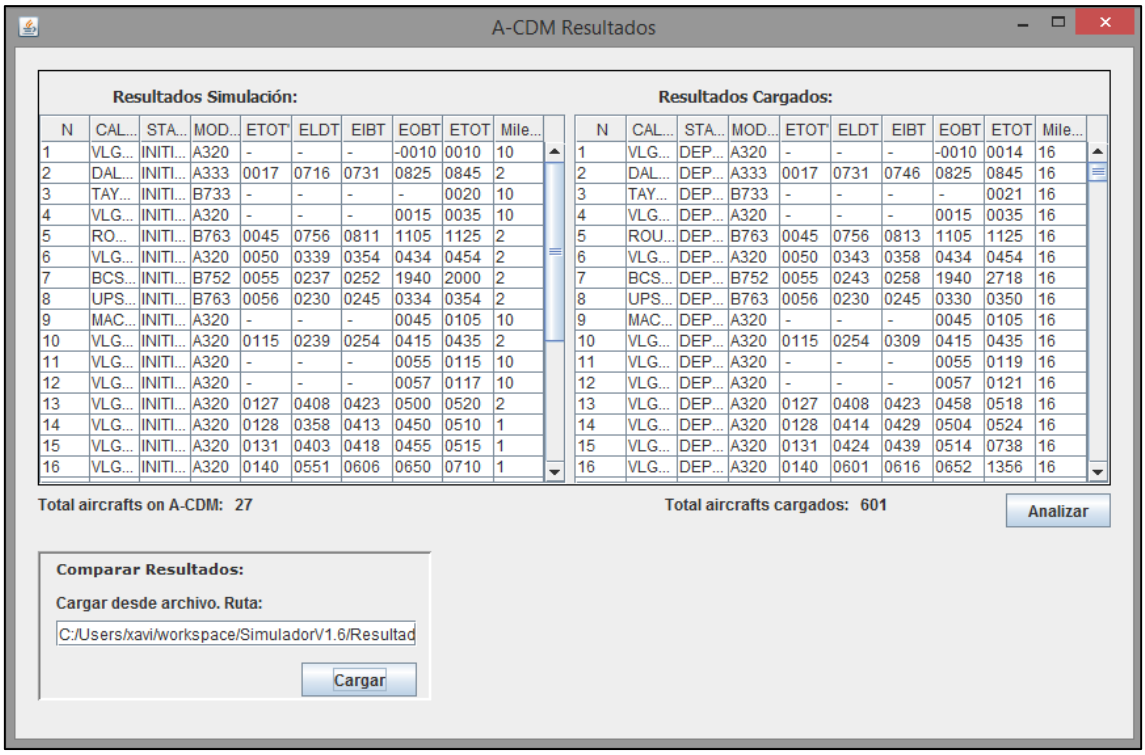


Figura D-44: Interfaz Resultados con cargados

Tano si hay una tabla o dos, el programa permite al usuario pasar a analizar los diferentes parámetros de los aviones. Para pasar a este análisis es necesario clicar en el botón de *Analizar*

D.7.4 Análisis

Al pulsar el botón *Analizar*, podemos ver cómo se nos abre la interfaz D-45

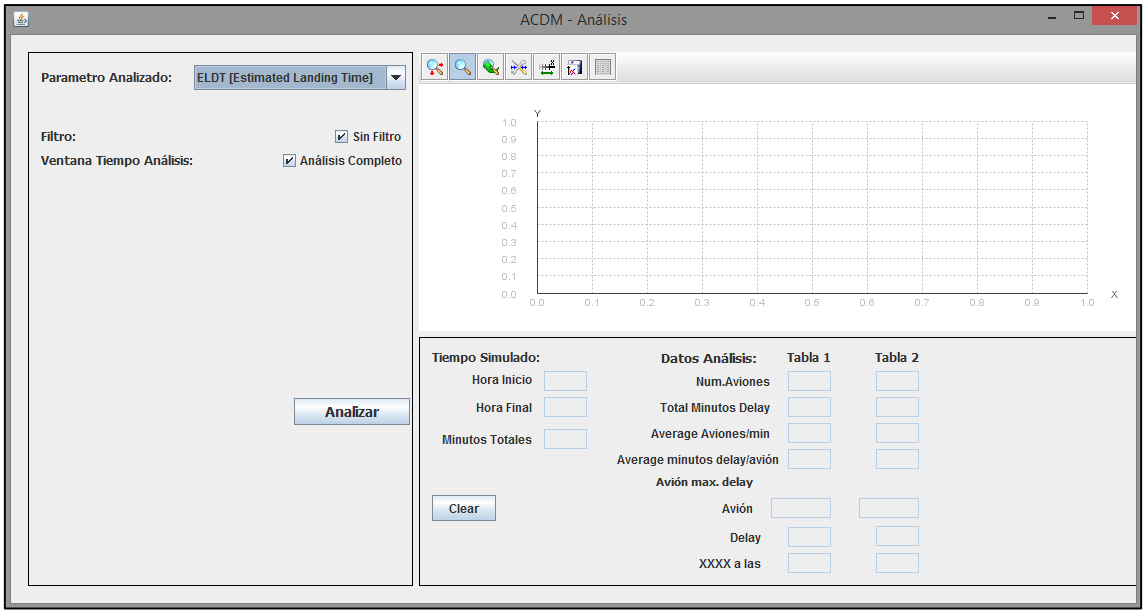


Figura D-45: Interfaz Análisis

En esta interfaz primero tenemos de seleccionar los parámetros sobre los que queremos realizar el análisis de los resultados. Estos parámetros se escojen en la parte izquierda de la pantalla.

Primero escojeremos el parámetro o tiempo del avión que queremos estudiar, entre los que ofrece el comboBox *Parametro Analizado*.

A continuación los filtros la interfaz da la posibilidad de añadir filtros a los aviones que se desean analizar, además de decidir acotar el análisis a una ventana de tiempo determinada.

Podemos decidir si tener en cuenta todos los aviones o solo los de una AO en concreto o los aviones de un mismo modelo, a la vez que podemos escojer el periodo de tiempo que estamos interesados en estudiar. Si no escojemos ningun filtro el programa entenderá que estudiamos todos los aviones durante todo el tiempo que ha durado la simulación, desde el primer avión hasta el último de todos.

Una vez escojidos los parámetros pulsaremos el botón *analizar*. El programa completará con los valores analizados todas las casillas de la parte inferior, y pintará la gráfica con los datos encontrados.

En la grafica podemos ver cómo el eje X indica el minuto de la simulación, y el eje Y indica la cantidad de aviones que están sufriendo un retraso en ese minuto. Si solo estamos analizando un documento los valores de la tabla 2 quedarán vacios.

En la figura D-46 podemos ver un ejemplo de cómo quedaria una interfaz despues de analizar unos resultados:

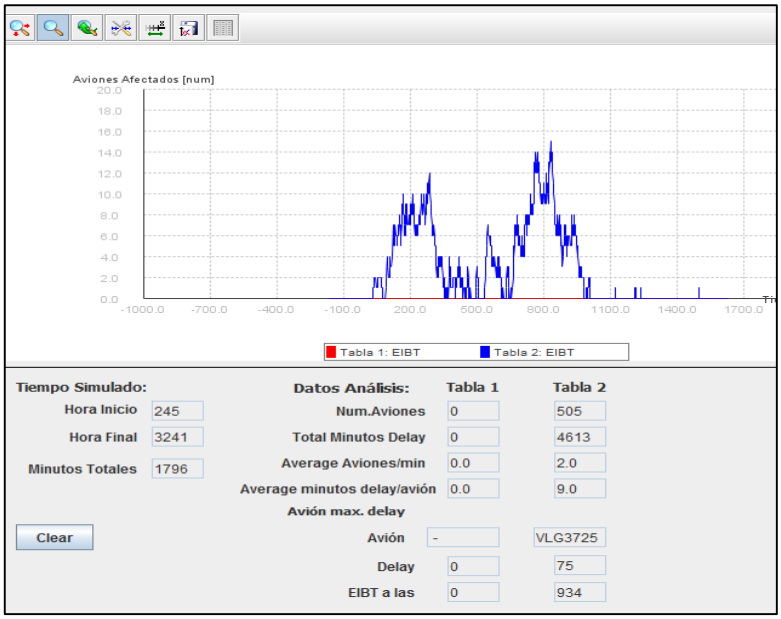


Figura D-46: Ejemplo Análisis

Con el botón *Clear* podemos limpiar la gráfica y los valores para estudiar otros parámetros. Si queremos analizar otro parámetro y compararlo con el se acaba de analizar, si no clicamos en *Clear* se mostrarán ambas graficas a la vez.

Esta gráfica tambien ofrece la posibilidad de cambiar ejes, hacer zoom y cambiar el color de las lineas de los datos.

APÉNDICE E. Tutorial JADE

E.1 Instalación

La versión de JADE utilizada en este proyecto ha sido la 3.2, puede bajarse directamente y de forma gratuita de la página web oficial del desarrollador, en esta dirección:

http://jade.tilab.com

Aunque se trata de una descarga gratuita, para obtenerla hay que rellenar una serie de formularios. La web ofrece una serie de archivos comprimidos *zip*, que son los siguientes:

- JADE-doc-3.2.zip: la documentación javadoc, el manual del administrado, el del programador y un tutorial.
- JADE-src-3.2.zip: el código fuente sin compilar.
- JADE-bin-3.2.zip: el código ya compilado y listo para ser interpretado
- JADE-examples-3.2.zip: ejemplos de uso de la plataforma.
- JADE-all-3.2.zip: Contiene todos los ficheros anteriores.

Después de obtener nuestra cuenta y haber rellenado los formularios, para la instalación solo es necesario descargar y descomprimir el ultimo archivo comprimido (JADE-all-3.2.zip).
Una vez tenemos la carpeta con los ficheros descomprimidos, solo es necesario crear un directorio dentro del proyecto en el que queramos utilizar JADE y copiar la carpeta entera ahí.

De esta forma tendremos, dentro de cada uno de los proyectos donde queramos utilizar JADE, una carpeta con todo lo descomprimido de JADE-all-3.2.zip.
Después de añadir la carpeta al proyecto solo es necesario añadir las librerías necesarias para que JADE se ejecute correctamente. Para ello es necesario utilizar los 3 archivos .jar que podemos encontrar en la carpeta *lib* y en *lib/commons-codec*. Los 3 .jar que hay que añadir son:

- *commons-codec-1.3.jar*
- *jade.jar*
- *jadeExamples.jar*

Para añadir estas librerías solo es necesario clicar con el botón derecho del ratón en cada uno de ellos y hacer *Build path→Add to build path.*
Una vez añadidas estas librerías podremos ejecutar cualquier programa de JADE sin problemas.

Es importante remarcar que hay que repetir estos pasos para cada uno de los proyectos que vayan a utilizar JADE.

E.2 Como ejecutarlo

Una de las peculiaridades de JADE es que no se comporta cómo un programa normal, no es una ejecución "línea a línea" pura, puesto que se trata de un programa descentralizado con comunicaciones asíncronas entre los agentes definidos. Por eso, a la hora de ejecutar JADE, el programa que ejecutamos es el llamado *jade.Boot*, este se encarga prepara el terreno y abrir los agentes que hayamos definido para la simulación.

Por tanto, en todas las simulaciones tendremos, aparte de los agentes que definamos nosotros, tres agentes que siempre se ejecutan: AMS, DF y RMA; estos agentes se encargan de administrar las comunicaciones y el *Enviroment* en el que se ejecutará el programa. A la hora de programar y simular, estos agentes no nos tienen que preocupar y no tenemos que tocarlos.

Para explicar cómo ejecutar un programa multiagente con JADE hemos utilizado un pequeño ejemplo. Concretamente hemos utilizado un par de ejemplos con agentes ya programados, que vienen incluidos en el archivo *zip* previamente descargado y añadido al proyecto.

E.2.1 Ejemplo de ejecución de JADE

Para comprobar si JADE funciona realizamos una prueba con alguno de los ejemplos que tenemos en la librería jadeExamples.jar

Lo más recomendado para probar los ejemplos es crear un nuevo pakcage en la carpeta src y añadirle las clases/agentes que se utilizaran en el ejemplo
Para esta primera prueba se utiliza el ejemplo "hello" para mostrar cómo se tiene que ejecutar correctamente JADE. En este ejemplo ejecutaremos dos agentes muy simples que lo único que hacen al ejecutarse es escribir por consola su nombre (Nick en el programa) y "hello world". Es un ejemplo sencillo pero no servirá para ver que además se pueden ejecutar dos agentes de la misma clase en un mismo programa, gracias a que hay que darle un nombre a cada uno.

Para la ejecución de este ejemplo, y de cualquier programa con JADE, es necesario ir a ***run → run config***, para cambiar diferentes parámetros que en un programa normal no hace falta.

Al abrir *run config* se nos abre una ventana con diferentes pestañas. Las únicas pestañas que nos interesan son las dos primeras:

- Pestaña *MAIN*:
  - Damos nombre a la ejecución
  - Escribimos el nombre del proyecto que vamos a ejecutar
  - Main: jade.Boot. Es la clase que se encargará de ejecutar todos nuestros agentes
  - Seleccionamos "*Include system libraries when searching for main class"*

- Pestaña ARGUMENTS:
  En esta pestaña añadimos los agentes que se ejecutaran en la simulación, así cómo una interfaz gráfica predefinida de JADE, muy útil para seguir la simulación. En el apartado *Program Arguments* añadimos
  - *–gui*: Para añadir la interfaz gráfica de JADE
  - *-agents: nick:ruta.nombreClase;* Para cada uno de los agentes que vayamos a tener en el programa.

  En este ejemplo ejecutaremos dos agentes que se encuentran en el Package “hello”, ambos son de la misma clase “HelloWorldAgent”. Por tanto, el apartado de *Program Arguments* quedará así:
  -gui -agents Pedro:hello.HelloWorldAgent;Ricardo:hello.HelloWorldAgent

Una vez tenemos las *run config* cómo queremos, ya se puede ejecutar el programa. En este caso, al ejecutar el programa con los parámetros de este ejemplo, se nos abre la ventana “-gui” mostrándonos los agentes que se están ejecutando, figura E-1:

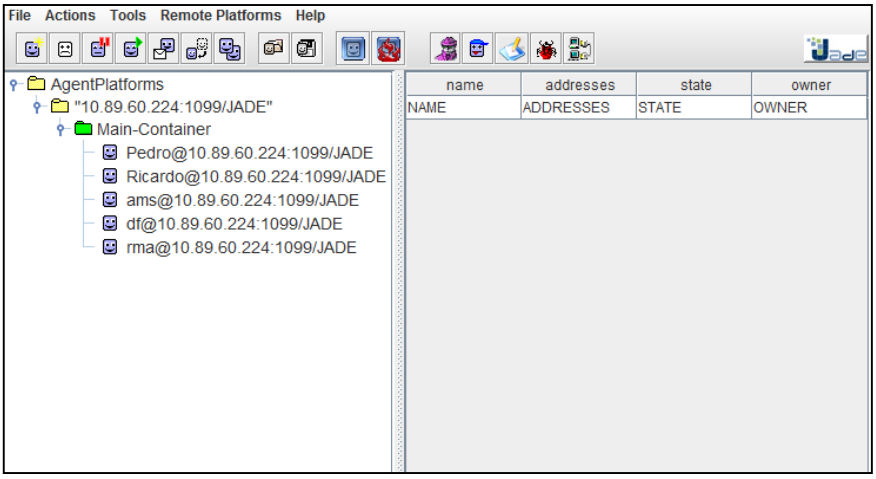


Figura E-1: Interfaz -gui JADE

Y en consola aparecen muchas líneas en rojo (figura E-2) que no son un error, sino que son predefinidas al ejecutar cualquier programa JADE; y los “hello World” de los dos agentes:

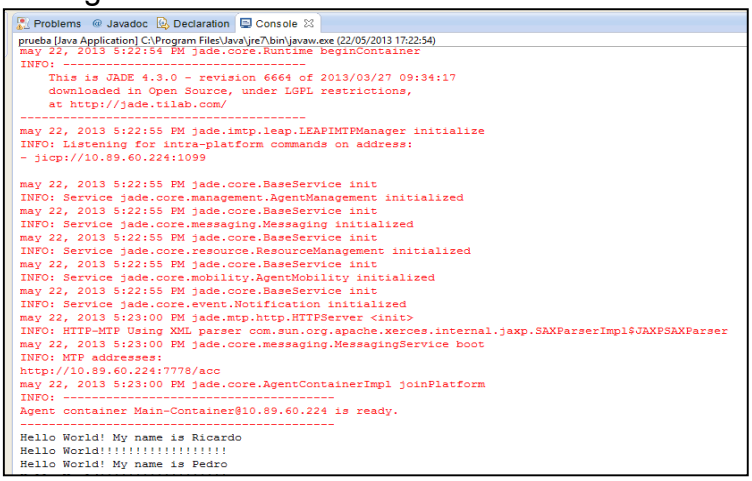


Figura E-2: Resultado Consola Ejemplo JADE

E.3 Interfaz de control/visualización

En este tutorial hemos hablado de una interfaz predefinida de JADE que nos permite seguir la simulación en tiempo real. Esta interfaz nos permite visualizar todos los agentes que tenemos ejecutándose en el programa, así cómo opciones varias para observar y actuar en la simulación.
En la figura E-3 podemos ver el menú principal. En este menú principal tenemos muchas opciones y botones disponibles, pero en este caso explicaremos las que han sido utilizadas para este proyecto y, bajo nuestro criterio, han sido muy útiles:

1. Actuaciones sobre agentes: Creación, parada, clonación y otras opciones a agentes
2. Sniffer: Espía de comunicaciones
3. Dummie Agent: Agente sencillo para pruebas

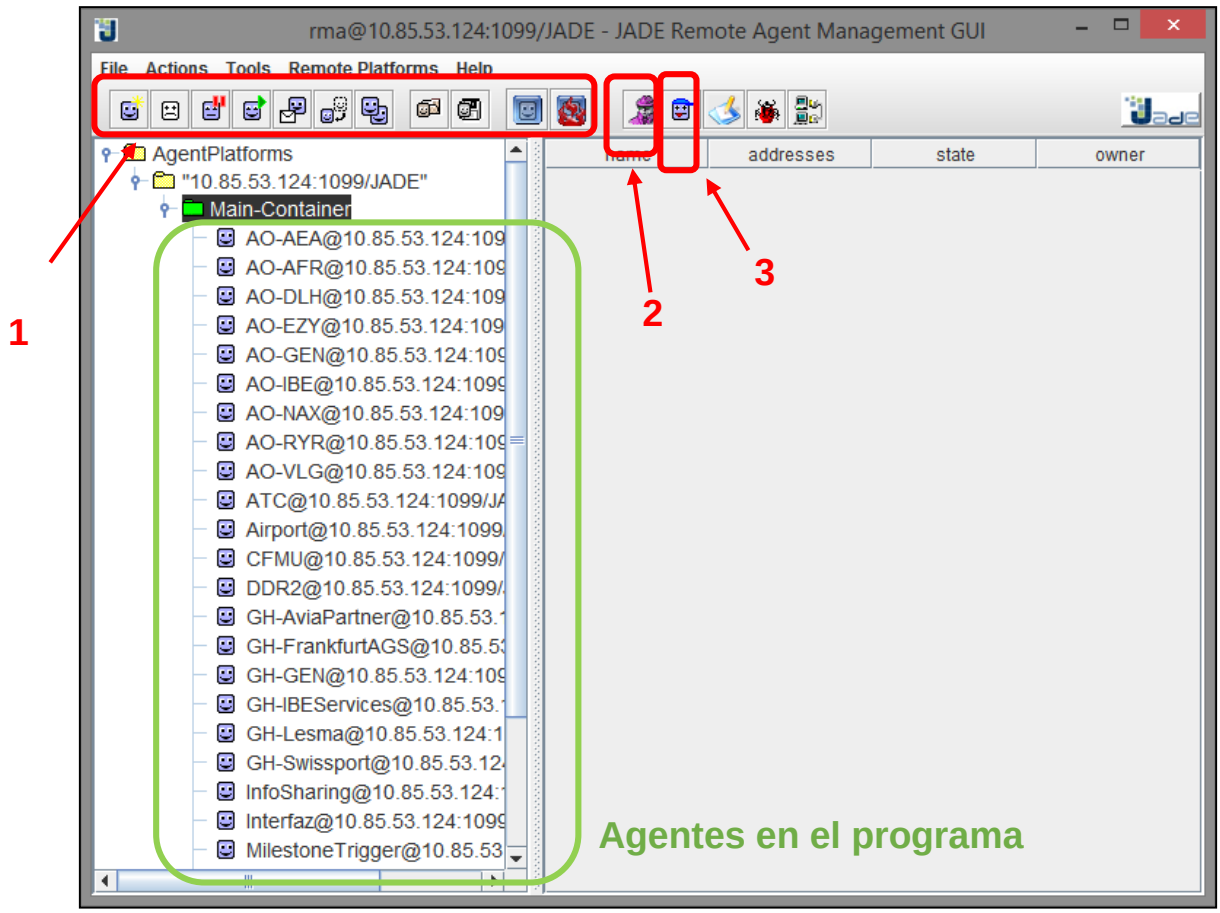


Figura E-3: Interfaz –gui JADE

E.3.1 Actuaciones sobre agentes

En el apartado 1 podemos encontrar, en el mismo orden, las opciones siguientes:

Crear Agente	Enviar Mensaje	Cargar Agente
Matar Agente	Migrar Agente	Freeze Agente
Pausar Agente	Clonar Agente	Thaw Agente
Reanudar Agente	Guardar Agente	

Estas opciones te permiten seguir definiendo y borrando agentes aunque ya este la simulación ejecutándose. También permitiría no rellenar el apartado *Program Parameters* de *Run Config* y definir todos los agentes de la simulación con las opciones “Crear Agente” y “Cargar Agente”.

E.3.2 Sniffer

Esta opción nos permite añadir un “espía” a nuestra simulación. Este espía se encargará de mostrarnos en una interfaz gráfica un diagrama con los mensajes que se envían durante la simulación cómo el de la figura E-4:

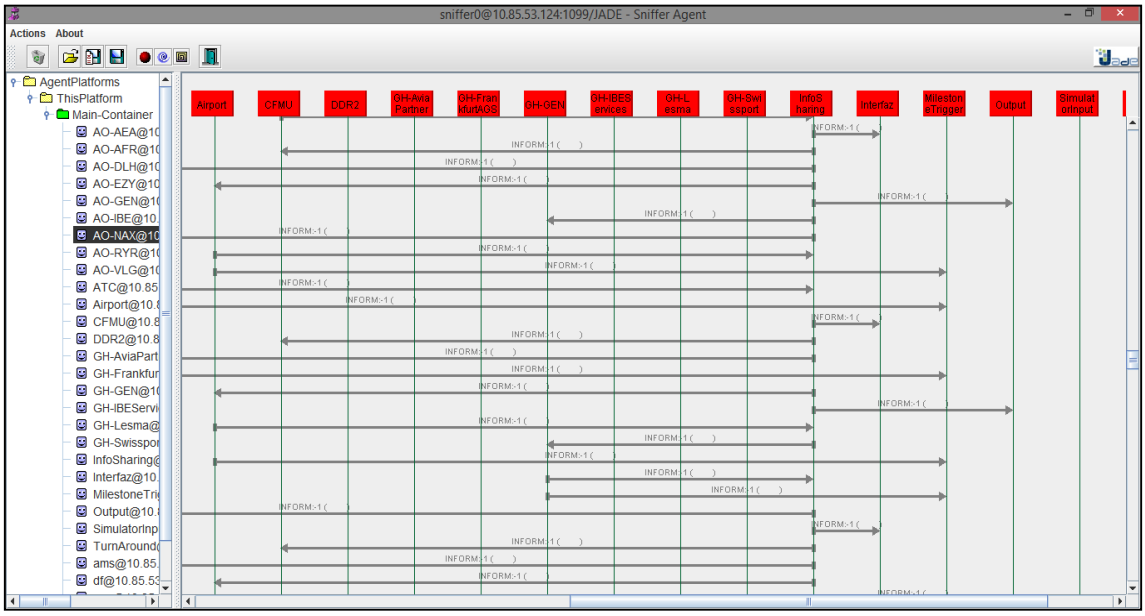


Figura E-4: Ejemplo Captura Comunicaciones Sniffer

Esta interfaz nos permite elegir que agentes queremos espiar, por tanto, antes de nada es necesario seleccionar esos agentes y darle al botón “Do sniff this agent(s)” (botón con un círculo rojo). En ese momento aparecen los agentes en la parte derecha de la pantalla, donde cada cuadro rojo representa un agente.

En la imagen previa podemos ver cómo muestra este sniffer los mensajes. Los mensajes están representados cómo flechas de un agente a otro, y están ordenados y numerados de forma temporal; cosa muy útil para saber el volumen de comunicaciones y la secuencia de mensajes de la simulación.

Estas flechas, que representan cada uno de los mensajes, nos permiten ver que contiene cada uno. Haciendo doble-click encima de cualquiera de ellas nos muestra los parámetros más importantes del mensaje; cómo su remitente, destinatario y contenido. En la figura E-5 se puede ver este menú.

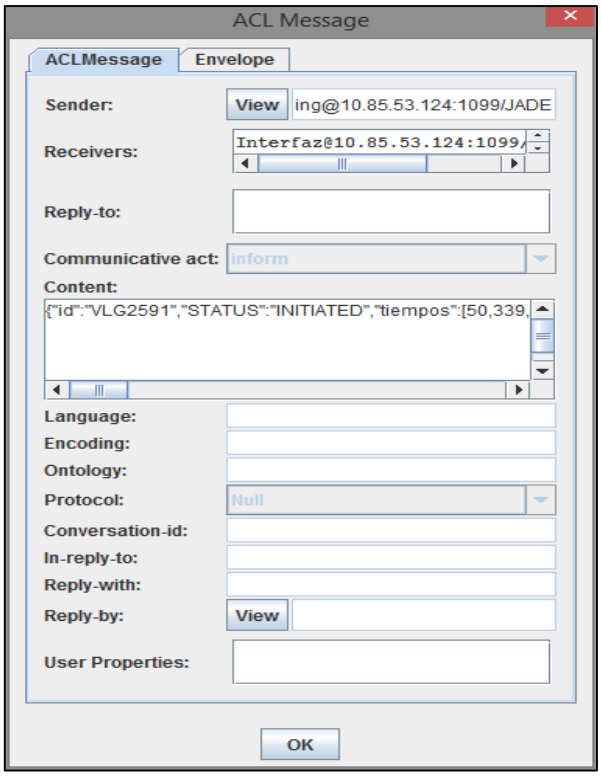


Figura E-5: Interfaz Mensaje Capturado

E.3.3 Dummie Agent

Esta opción nos permite crear un agente “dummie”, un agente que no hemos programado nosotros y nos sirve para hacer diferentes pruebas.
Este agente se utiliza para hacer pruebas de comunicación y comportamientos de reacción a mensajes de los diferentes agentes. Con una interfaz sencilla, figura E-6, nos permite rellenar los diferentes parámetros que contiene un mensaje.

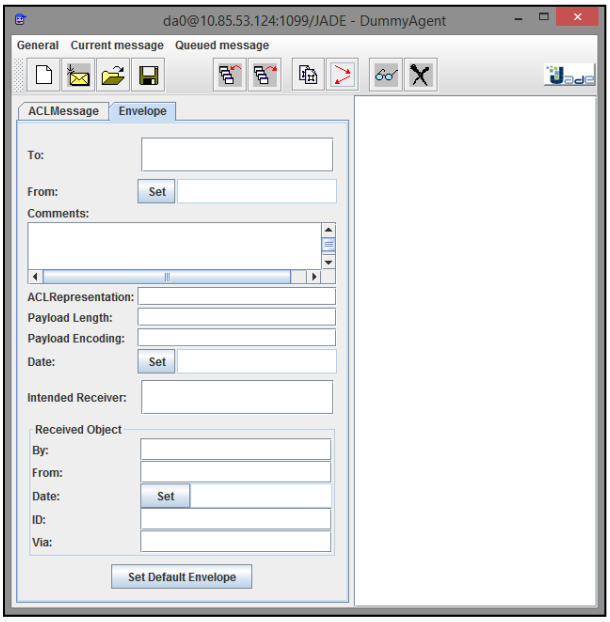


Figura E-6: Interfaz Dummie Agent

Estos mensajes se pueden utilizar para ver cómo los agentes se comportan a determinados mensajes, por ejemplo se puede probar a enviar un mensaje con información errónea o mensajes que se envían en casos muy especiales y que en una simulación normal serian difíciles de ver.

Este agente también nos permite cargar un mensaje que hayamos guardado previamente y programar una respuesta rápida en caso de que tenga que recibir un mensaje de alguno de los agentes del programa.

APÉNDICE F. Métodos de cada Agente

F.1 Introducción

En este apéndice se mostrará, de manera esquematizada, una explicación de cada uno de los métodos y funciones que realiza cada agente. Está pensado para revisarse a la vez que el código del programa, cómo ayuda a la comprensión de este, aunque el código también ha sido comentado.

Este apéndice no hubiera sido necesario en un TFG normal, pero debido a que este proyecto está pensado para que otros alumnos lo continúen, se ha realizado este apartado para ayudar a la comprensión del código.

Por tanto, solo se recomienda la lectura de este apéndice a aquellos que quieran comprender el código a nivel de programador, para trabajar en él para mejorarlo o añadirle nuevas funcionalidades al programa.

F.2 Métodos Agentes

F.2.1 Métodos CDM

Función:	Actuamos
Quien la llama:	- Agente Airport - Agente CFMU - SimuladorGrafica
Datos de entrada:	- Aircraft Avión - int[ ] opción
Actuación:	La función comprueba si el avión de entrada ha actualizado alguno de los valores de la lista opción.
Funciones a las que llama:	- comprobarActualizadas
Datos Salida:	- int: 1→ Devuelve 1 si algún elemento de opción coincide con alguno de Actualizadas del avión. 2→ Devuelve 0 si ningún elemento de opción coincide con ninguno de Actualizadas del avión.

Función:	comprobarActualizadas
Quien la llama:	- función Actuamos - Agente AO - Agente CDM - Agente GH
Datos de entrada:	- Aircraft Avión - int[ ] opción
Actuación:	La función comprueba si el avión de entrada ha actualizado alguno de los valores de la lista opción.
Funciones a las que llama:	-

Datos Salida:	- boolean: true→ Devuelve true si algún elemento de opción coincide con alguno de Actualizadas del avión. False→ Devuelve false si ningún elemento de opción coincide con ninguno de Actualizadas del avión.

Función:	PublicarInformacion
Quien la llama:	- Agente Airport - Agente ATC - Agente CFMU - Agente GH - Agende AO
Datos de entrada:	- Aircraft Avión - int muestralo
Actuación:	Es la función a la que recurriremos cada vez que queramos avisar al IS de alguna actualización. Con el valor muéstralo podemos decidir si queremos mostrar alguna información por pantalla.
Funciones a las que llama:	-
Datos Salida:	-

Función:	RecepcionMensaje
Quien la llama:	- Todos los agentes que reciben mensajes en algún momento de la simulación
Datos de entrada:	- ACLMensaje msg - int muestralo
Actuación:	Decodifica el mensaje ACL a información comprendida dentro de una clase Aircraft. Muestralo indica si se muestra la información por pantalla
Funciones a las que llama:	- ExtraerInfo
Datos Salida:	- Aircraft avion

Función:	ExtraerInfo
Quien la llama:	- Función RecepcionMensaje
Datos de entrada:	- ACLMensaje msg
Actuación:	Decodifica el mensaje ACL a información comprendida dentro de una clase Aircraft.
Funciones a las que llama:	-
Datos Salida:	- Aircraft avion

Función:	ExtraerInfoResults
Quien la llama:	- Agente Output
Datos de entrada:	- String msg: es la información ACL de un mensaje comprendida en un String
Actuación:	Decodifica el mensaje String a información comprendida dentro de una clase Aircraft.
Funciones a las que llama:	-
Datos Salida:	- Aircraft avion

Función:	restatemps
Quien la llama:	- Cualquiera que desee restar dos tiempos
Datos de entrada:	- int A - int B
Actuación:	A menos B, teniendo en cuenta hhmm
Funciones a las que llama:	-
Datos Salida:	- int C: resultado de la resta

Función:	sumatemps
Quien la llama:	- Cualquiera que desee sumar dos tiempos
Datos de entrada:	- int A - int min : no es una hora, es cantidad de minutos a sumar
Actuación:	Suma los minutos min al valor A, teniendo en cuenta hhmm
Funciones a las que llama:	-
Datos Salida:	- int C: resultado de la suma

Función:	Calculotiempo
Quien la llama:	- Cualquiera que desee sumar o restar dos tiempos
Datos de entrada:	- int tiempo - int delay : no es una hora, es cantidad de minutos a sumar o restar - int operación: 1→sumar, 0→restar
Actuación:	Suma o resta los minutos delay al valor tiempo, teniendo en cuenta hhmm
Funciones a las que llama:	-
Datos Salida:	- int result: resultado de la suma

Función:	TempsQueRetrasem
Quien la llama:	- Agente Airport - Agente GH - Agente CFMU - Agente ATC - Agente AO
Datos de entrada:	- int X
Actuación:	Para cada valor de entrada X encuentra cual es el tiempo que se retrasará en los retrasos. X equivale al valor de “Situación” en los retrasos tipo 1 y tipo 2.

Funciones a las que llama:	-
Datos Salida:	- String TiempoVar

Función:	MirarmodeliAO
Quien la llama:	- Agente Airport - Agente GH - Agente CFMU - Agente ATC - Agente AO
Datos de entrada:	- Aircraft avion - String AOs - String Modelos
Actuación:	Comprueba si el modelo del avión está dentro de los Modelos, y la AO dentro de la lista AOs de entrada
Funciones a las que llama:	-
Datos Salida:	- boolean correctos

Función:	AlgunRetraso
Quien la llama:	- Agente Airport - Agente GH - Agente CFMU - Agente ATC - Agente AO
Datos de entrada:	- String Id - int Milestone - ArrayList Retrasoavion
Actuación:	Busca si la lista Retrasoavion tiene guardado una información correspondiente al avión con callsign=Id en el momento de Milestone.
Funciones a las que llama:	- TreureMilestone
Datos Salida:	- String[2] Retraso: El primer valor contiene información del retraso a aplicar, el segundo informa del indicie de la lista Retrasoavion donde se ha encontrado la infomación

Función:	DetectarRetrasoTiempo
Quien la llama:	- Agente Airport - Agente GH - Agente CFMU - Agente ATC - Agente AO
Datos de entrada:	- String hora - ArrayList Retrasotiempo
Actuación:	Busca si la lista Retrasotiempo tiene guardado una información correspondiente a un retraso para la hora en la que nos encontramos
Funciones a las que llama:	-

Datos Salida:	- String Retraso: Devuelve un número, que contiene el índice en la lista Retrasotiempo donde encontraremos la información del retraso que tendremos de aplicar.

Función:	TrobarRetraso
Quien la llama:	- Agente ATC
Datos de entrada:	- String Hora - ArrayList Retrasopista
Actuación:	Busca si la lista Retrasopista tiene guardado una información correspondiente a un retraso para la hora en la que nos encontramos
Funciones a las que llama:	-
Datos Salida:	String retraso: devuelve la la información del retraso que tendremos de aplicar debido a una reducción de la capacidad de la pista

Función:	GetTempsPrevMil
Quien la llama:	- Agente Airport - Agente GH - Agente CFMU - Agente ATC - Agente AO
Datos de entrada:	- int a
Actuación:	Recibe el valor de un milestone e indica cual es el tiempo previo
Funciones a las que llama:	-
Datos Salida:	- int Tiempo

Función:	GetTempsPostMil
Quien la llama:	- Agente Airport - Agente GH - Agente CFMU - Agente ATC - Agente AO
Datos de entrada:	- int a
Actuación:	Recibe el valor de un milestone e indica cual es el tiempo posterior
Funciones a las que llama:	-
Datos Salida:	- int Tiempo

Función:	GetTempsActualMil
Quien la llama:	- Agente Airport - Agente GH - Agente CFMU - Agente ATC - Agente AO
Datos de entrada:	- Aircraft avión - int a

Actuación:	Recibe el valor de un milestone correspondiente a un avión. Busca la hora en la que sucede ese milestone para ese avión.
Funciones a las que llama:	-
Datos Salida:	- int Tiempo

Función:	TreureMilestone
Quien la llama:	-función AlgunRetraso
Datos de entrada:	- String Temps
Actuación:	Recibe el nombre de un tiempo i devuelve el milestones con el que está relacionado
Funciones a las que llama:	-
Datos Salida:	- int valor

Función:	TreureTemps
Quien la llama:	- Agente Airport - Agente GH - Agente CFMU - Agente ATC - Agente AO
Datos de entrada:	- Aircraft a - String Temps
Actuación:	Recibe un avión y la información del tiempo que tiene de extraer del avión.
Funciones a las que llama:	-
Datos Salida:	- int valor

Función:	TreureActualizada
Quien la llama:	- Agente Airport - Agente GH - Agente CFMU - Agente ATC - Agente AO
Datos de entrada:	- String Temps
Actuación:	Recibe un tiempo y busca cual es el numero para el vector de actualizadas
Funciones a las que llama:	-
Datos Salida:	- int[2] a : el primer valor es el que corresponde al tiempo actualizado, el segundo es un 19, que indica que se han actualizado las actializadas.

Función:	takeDown
Quien la llama:	- Lo llaman los agentes al tener algún error, para avisar de que se cierran.
Datos de entrada:	-
Actuación:	- se prepara para mandar un mensaje para avisar de que algún agente se ha cerrado
Funciones a las que llama:	- maydaymayday
Datos Salida:	-

Función:	maydaymayday
Quien la llama:	- funcion takeDown
Datos de entrada:	-
Actuación:	- manda un mensaje para avisar de que algún agente se ha cerrado
Funciones a las que llama:	-
Datos Salida:	-

Función:	mensajeMilestone
Quien la llama:	- Agente Airport - Agente GH - Agente CFMU - Agente ATC - Agente AO
Datos de entrada:	- Aircraft Avion - String[ ] Milestone - int Opcion
Actuación:	Envía un mensaje al MilestoneTrigger para avisar de que se ha actualizado algún milestone de la lista
Funciones a las que llama:	-
Datos Salida:	-

Función:	AplicarNormal
Quien la llama:	- Agente GH - Agente ATC - Agente AO
Datos de entrada:	- int variacio - boolean variacionesnormales
Actuación:	Calcula el valor del delay a añadir al tiempo siguiendo una probabilidad normal
Funciones a las que llama:	-
Datos Salida:	- int Delay

Función:	ValorParametro
Quien la llama:	- Cualquier agente
Datos de entrada:	- String parametro
Actuación:	Recoje el valor del parámetro del avionParametros (Inout: línea parámetro ejemplo “Aviones 12”)
Funciones a las que llama:	-
Datos Salida:	- String valor: Devuelve el segundo elemento de parámetro después de hacer un split de espacio.

Función:	sacarParametros (1)
Quien la llama:	- Cualquier agente
Datos de entrada:	- Aircraft avion
Actuación:	Saca los parámetros de las incidencias de avion y los devuelve en forma de vector de Strings

Funciones a las que llama:	-
Datos Salida:	- String[ ] listaparametros

Función:	sacarParametros (2)
Quien la llama:	- Cualquier agente - int posicion
Datos de entrada:	- Aircraft avion
Actuación:	Saca los parámetros de las incidencias de avion y devuelve el que se encuentra en el índice posicion
Funciones a las que llama:	-
Datos Salida:	- String parametro

Función:	sacarParametros (2)
Quien la llama:	- Cualquier agente - int posicion
Datos de entrada:	- Aircraft avion
Actuación:	Saca los parámetros de las incidencias de avion y devuelve el que se encuentra en el índice posicion
Funciones a las que llama:	-
Datos Salida:	- String parametro

Función:	listaaerolineas
Quien la llama:	- Agente AO - Agente CDM - Agente MilestoneTrigger - Agente InfoSharing
Datos de entrada:	- String parámetros[ ]
Actuación:	Devuelve un lista de Strings, en cada palabra contiene el nombre de un agente AO.
Funciones a las que llama:	-
Datos Salida:	- String aerolineas [ ]

F.2.2 Métodos agente Infosharing

Función:	DecidirGH
Quien la llama:	- Comportamiento ActualizarTabla al hacer broadcast de alguna información.
Datos de entrada:	- Aircraft avion - int GHamount - String AOforGH
Actuación:	Busca el agente GH que tiene el contrato con la aerolínea del avión de entrada.
Funciones a las que llama:	-
Datos Salida:	- AID[ ] CDMAgents

Función:	ActualizarTabla
Quien la llama:	- Comportamiento ActualizarTabla
Datos de entrada:	- Aircraft mensaje

	- Aircraft avionTabla - int muestralo
Actuación:	Actualiza el avionTabla según las actualizadas del mensaje.
Funciones a las que llama:	- actualizarIncidencias
Datos Salida:	- Aircraft avionTabla

Función:	BroadcastInfo
Quien la llama:	- Comportamiento ActualizarTabla
Datos de entrada:	- Aircraft avion - int muestralo - AID[ ] Agentes
Actuación:	Envía un mensaje con la información del avion a los Agentes receptores
Funciones a las que llama:	- AerolineaAvion
Datos Salida:	-

Función:	AerolineaAvion
Quien la llama:	- BroadcastInfo
Datos de entrada:	- Aircraft avion - String[ ] aerolineas
Actuación:	Busca el agente AO que se ocupará del avión. Si no existe un agente AO para la aerolínea del avión será de AO- GEN
Funciones a las que llama:	-
Datos Salida:	- AID Aerolinea

Función:	Enviamosinfo_SioNO
Quien la llama:	- Comportamiento ActualizarTabla
Datos de entrada:	- int posicion
Actuación:	Mira si el avión de la tabla situado en posición y si ha actualizado alguna información realiza el broadcast.
Funciones a las que llama:	-
Datos Salida:	- boolean enviamos

Función:	SetMessages
Quien la llama:	- Comportamiento RecepcinMessage
Datos de entrada:	- ACLMessage text - ArrayList ListaMensajes
Actuación:	Añade un mensaje a la ListaMensajes de manera que entre comportamientos se puedan transferir informaciones.
Funciones a las que llama:	-
Datos Salida:	- ArrayList ListaMensajes

Función:	GetMessages
Quien la llama:	- Comportamiento ActualizarTabla
Datos de entrada:	- ArrayList ListaMensajes

Actuación:	Coge un mensaje a la ListaMensajes de manera que entre comportamientos se puedan transferir informaciones.
Funciones a las que llama:	-
Datos Salida:	- ArrayList ListaMensajes - ACLMessage text

Función:	actualizarIncidencias
Quien la llama:	- ActualizarTabla: Si se actualizan las incidencias, actualizada=19
Datos de entrada:	- String inTabla - String inMensaje
Actuación:	Añade las incidencias del mensaje a las de la Tabla, teniendo en cuenta que si una incidencia ya existía no la tiene de añadir.
Funciones a las que llama:	-
Datos Salida:	- String incidencias

F.2.3 Métodos agente AO

Función:	cargarAvionesCompañia
Quien la llama:	- Comportamiento RecepcionAvion cuando llega un mensaje con OpcionMensaje=1
Datos de entrada:	-
Actuación:	Función que llama a la función que leerá el documento VuelosAerolinea.txt
Funciones a las que llama:	- leerArchivoAO
Datos Salida:	- String[ ][ ] avionesCompañia

Función:	leerArchivoAO
Quien la llama:	- cargarAvionesCompañia
Datos de entrada:	- String ubicacion
Actuación:	función que lee el documento VuelosAerolinea.txt
Funciones a las que llama:	- aerolineagenerica
Datos Salida:	- String[ ][ ] avionesCompañia

Función:	relacionarFP
Quien la llama:	- Comportamiento RecepcionAvion cuando intenta definir el ETOT de un avión nuevo en la tabla del agente
Datos de entrada:	- String[ ][ ] avionesCompañia int posicion: posición de un avión en la matriz avionesCompañia
Actuación:	Mira si el avión que se encuentra en posición es de solo llegada o no.
Funciones a las que llama:	-
Datos Salida:	- boolean sololllega

Función:	aerolineagenerica
Quien la llama:	- leerArchivoAO
Datos de entrada:	- String[ ] datos: información de un avión - String[ ] aerolíneas: lista de aerolíneas con agente propio
Actuación:	Mira si el avión de datos pertenece a la aerolínea genérica o pertenece a una aerolínea con agente AO propio
Funciones a las que llama:	-
Datos Salida:	- boolean generic

Función:	PonerRetraso
Quien la llama:	- Comportamiento RecepcionAvion cuando llega un mensaje con OpcionMensaje=4
Datos de entrada:	- String Retraso
Actuación:	Le llega un retraso y lo guarda en la lista Retrasotiempo(tipo 1) o Retrasoavion (tipo 2 y 3)
Funciones a las que llama:	- Tempsqueretrasem (CDM)
Datos Salida:	- ArrayList Retrasotiempo - ArrayList Retrasoavion

Función:	BuscarRetrasosTiempos
Quien la llama:	- Comportamiento RecepcionAvion cuando llega un mensaje con OpcionMensaje=5
Datos de entrada:	- String posicio - int encontrados - ArrayList Retrasotiempo
Actuación:	Encuentra todos los aviones afectados por un retraso de tipo 1, y escribe los retrasos cómo tipo 3
Funciones a las que llama:	- GetTempsPreviMil (CDM) - GetTempsPostMil (CDM) - restatemps (CDM) - MirarmodeliAO (CDM)
Datos Salida:	- String LlistaRetrasos - ArrayList Retrasotiempo

Función:	AplicarRetras
Quien la llama:	- Comportamiento RecepcionAvion cada vez que tiene de aplicar un retraso.
Datos de entrada:	- String[ ] Informació : información de un retraso tipo 3 - String hora
Actuación:	Aplica un retraso de tipo 3 a cualquier avión
Funciones a las que llama:	- TreureTemps (CDM) - TreureActualizada (CDM) - sumatemps (CDM) - PublicarInformacion (CDM)
Datos Salida:	-

F.2.4 Métodos agente GH

Función:	PonerRetraso
Quien la llama:	- Comportamiento RecepcionAvion cuando llega un mensaje con OpcionMensaje=4
Datos de entrada:	- String Retraso
Actuación:	Le llega un retraso y lo guarda en la lista Retrasotiempo(tipo 1) o Retrasoavion (tipo 2 y 3)
Funciones a las que llama:	- Tempsqueretrasem (CDM)
Datos Salida:	- ArrayList Retrasotiempo - ArrayList Retrasoavion

Función:	BuscarRetrasosTiempos
Quien la llama:	- Comportamiento RecepcionAvion cuando llega un mensaje con OpcionMensaje=5
Datos de entrada:	- String posicio - int encontrados - ArrayList Retrasotiempo
Actuación:	Encuentra todos los aviones afectados por un retraso de tipo 1, y escribe los retrasos cómo tipo 3
Funciones a las que llama:	- GetTempsPreviMil (CDM) - GetTempsPostMil (CDM) - restatemps (CDM) - MirarmodeliAO (CDM)
Datos Salida:	- String LlistaRetrasos - ArrayList Retrasotiempo

Función:	AplicarRetras
Quien la llama:	- Comportamiento RecepcionAvion cada vez que tiene de aplicar un retraso.
Datos de entrada:	- String[ ] Informació : información de un retraso tipo 3 - String hora
Actuación:	Aplica un retraso de tipo 3 a cualquier avión
Funciones a las que llama:	- TreureTemps (CDM) - TreureActualizada (CDM) - sumatemps (CDM) - PublicarInformacion (CDM)
Datos Salida:	-

Función:	AgenteExternoTurnAround
Quien la llama:	- estudioRotacion
Datos de entrada:	- Aircraft avion
Actuación:	Manda un mensaje al agente TurnAround pidiendo información sobre el tiempo de turn around de un avión.
Funciones a las que llama:	-
Datos Salida:	-

Función:	InformarTurnAround
Quien la llama:	- Comportamiento RecepcionAvion cuando le llega un mensaje dándole los parámetros de simulación.
Datos de entrada:	- Aircraft avion
Actuación:	Manda un mensaje al agente TurnAround para informarle de los parámetros de simulación. Este mensaje solo lo manda GH- GEN
Funciones a las que llama:	-
Datos Salida:	-

Función:	getAvionesInicio
Quien la llama:	Comportamiento RecepcionAvion.
Datos de entrada:	- ArrayList AvionesInicio: Lista de aviones, donde cada Aircraft contiene la información que ha llegado en un mensaje y no se ha podido procesar porque el agente estaba ocupado
Actuación:	- Mira si tenemos mensajes por procesar en la lista AvionesInicio, si no están todos procesados, coge uno y lo procesa.
Funciones a las que llama:	-
Datos Salida:	- Aircraft a - ArrayList AvionesInicio

Función:	ListaInicio
Quien la llama:	Comportamiento RecepcionAvion.
Datos de entrada:	- ArrayList AvionesInicio - Aircraft avion
Actuación:	- Si el agente recibe un mensaje, pero está ocupado procesando otro, guarda ese mensaje en AvionesInicio
Funciones a las que llama:	-
Datos Salida:	- ArrayList AvionesInicio

Función:	estudioRotacion
Quien la llama:	- Comportamiento RecepcionAvion cada vez que quiere comprobar que se respete el tiempo de turn around
Datos de entrada:	- Aircreaft Avio
Actuación:	Asegura que el tiempo entre EIBT i EOBT sea igual o mayor al mínimo tiempo necesario de turn around de cada aeronave
Funciones a las que llama:	- AgenteExternoTurnAround
Datos Salida:	- int[ ] retorna: primer valor EOBT, segundo valor 0→ EOBT no actualizado, 1→ EOBT actualizado

F.2.5 Métodos agente Airport

Función:	PonerRetraso
Quien la llama:	- Comportamiento RecepcionAvion cuando llega un mensaje con OpcionMensaje=4
Datos de entrada:	- String Retraso
Actuación:	Le llega un retraso y lo guarda en la lista Retrasotiempo(tipo 1) o Retrasoavion (tipo 2 y 3)
Funciones a las que llama:	- Tempsqueretrasem (CDM)
Datos Salida:	- ArrayList Retrasotiempo - ArrayList Retrasoavion

Función:	BuscarRetrasosTiempos
Quien la llama:	- Comportamiento RecepcionAvion cuando llega un mensaje con OpcionMensaje=5
Datos de entrada:	- String posicio - int encontrados - ArrayList Retrasotiempo
Actuación:	Encuentra todos los aviones afectados por un retraso de tipo 1, y escribe los retrasos cómo tipo 3
Funciones a las que llama:	- GetTempsPreviMil (CDM) - GetTempsPostMil (CDM) - restatemps (CDM) - MirarmodeliAO (CDM)
Datos Salida:	- String LlistaRetrasos - ArrayList Retrasotiempo

Función:	AplicarRetras
Quien la llama:	- Comportamiento RecepcionAvion cada vez que tiene de aplicar un retraso.
Datos de entrada:	- String[ ] Informació : información de un retraso tipo 3 - String hora
Actuación:	Aplica un retraso de tipo 3 a cualquier avión
Funciones a las que llama:	- TreureTemps (CDM) - TreureActualizada (CDM) - sumatemps (CDM) - PublicarInformacion (CDM)
Datos Salida:	-

Función:	CalculoEIBT
Quien la llama:	- Comportamiento RecepcionAvion cuando tiene de calcular EIBT
Datos de entrada:	- Aircraft a - int TaxiInTime
Actuación:	Calcula el EIBT de un avión en función de su ELDT y del tiempo de taxi- in
Funciones a las que llama:	-
Datos Salida:	- Aircraft a

F.2.6 Métodos agente CFMU

Función:	PonerRetraso
Quien la llama:	- Comportamiento RecepcionAvion cuando llega un mensaje con OpcionMensaje=4
Datos de entrada:	- String Retraso
Actuación:	Le llega un retraso y lo guarda en la lista Retrasotiempo(tipo 1) o Retrasoavion (tipo 2 y 3)
Funciones a las que llama:	- Tempsqueretrasem (CDM)
Datos Salida:	- ArrayList Retrasotiempo - ArrayList Retrasoavion

Función:	BuscarRetrasosTiempos
Quien la llama:	- Comportamiento RecepcionAvion cuando llega un mensaje con OpcionMensaje=5
Datos de entrada:	- String posicio - int encontrados - ArrayList Retrasotiempo
Actuación:	Encuentra todos los aviones afectados por un retraso de tipo 1, y escribe los retrasos cómo tipo 3
Funciones a las que llama:	- GetTempsPreviMil (CDM) - GetTempsPostMil (CDM) - restatemps (CDM) - MirarmodeliAO (CDM)
Datos Salida:	- String LlistaRetrasos - ArrayList Retrasotiempo

Función:	AplicarRetras
Quien la llama:	- Comportamiento RecepcionAvion cada vez que tiene de aplicar un retraso.
Datos de entrada:	- String[ ] Informació : información de un retraso tipo 3 - String hora
Actuación:	Aplica un retraso de tipo 3 a cualquier avión
Funciones a las que llama:	- TreureTemps (CDM) - TreureActualizada (CDM) - sumatemps (CDM) - PublicarInformacion (CDM)
Datos Salida:	-

Función:	CargarFP
Quien la llama:	- Comportamiento RecepcionAvion al recibir el mensaje de empezar la simulación, OpcionMensaje=1
Datos de entrada:	-String rutaTXT -StringrutaTXTS
Actuación:	Creamos la tabla con todos los aviones de los planes de vuelo en la memoria del agente

Funciones a las que llama:	- leerArchivoCDM - leerArchivoCDMSalidas
Datos Salida:	-String[ ][ ] FlightPlans -String[ ][ ] FlightPlansSalidas

Función:	CargarCat
Quien la llama:	- Comportamiento RecepcionAvion al recibir los parámetros de simulación, OpcionMensaje=3
Datos de entrada:	- String[ ] parametros
Actuación:	Escribe la lista ModelosCat i NumCat a partir de los datos de entrada (leídos del documento airport.txt)
Funciones a las que llama:	-
Datos Salida:	- ArrayList ModelosCat - ArrayList NumCat

Función:	CategoriaAvion
Quien la llama:	- IniciarCDM - IniciarCDMSalidas
Datos de entrada:	- Aircraft avion - ArrayList ModelosCat - ArrayList NumCat
Actuación:	Mira el modelo del avión, y busca su categoría en las listas ModelosCat y NumCat, si no encuentra su categoría le assigna un valor de 4.
Funciones a las que llama:	-
Datos Salida:	- int categoria

Función:	leerArchivoCFMUSalidas
Quien la llama:	- cargarFP
Datos de entrada:	- String ubicacion
Actuación:	Leer el documento FlightPlansSalidas.txt y devuelve la información en formato String[ ][ ]
Funciones a las que llama:	-
Datos Salida:	- String[ ][ ] FP

Función:	leerArchivoCFMU
Quien la llama:	- cargarFP
Datos de entrada:	- String ubicacion
Actuación:	Leer el documento FlightPlans.txt y devuelve la información en formato String[ ][ ]
Funciones a las que llama:	-
Datos Salida:	- String[ ][ ] FP

Función:	IniciarCDMSalidas
Quien la llama:	- Comportamiento RecepcionAvion al recibir un avión que solo sale, OpcionMensaje=6
Datos de entrada:	- String id - String[ ][ ] FP
Actuación:	Busca en FP un avión con una id concreta y crea el objeto Aircraft
Funciones a las que llama:	- CategoriaAvion
Datos Salida:	- Aircraft avionCDM

Función:	IniciarCDM
Quien la llama:	- Comportamiento RecepcionAvion al recibir un avion que solo entra o que realiza todo el proceso en nuestro aeropuerto(aterrizaje y despegue)
Datos de entrada:	- String id - String[ ][ ] FP
Actuación:	Busca en FP un avión con una id concreta y crea el objeto Aircraft
Funciones a las que llama:	- CategoriaAvion
Datos Salida:	- Aircraft avionCDM

F.2.7 Métodos agente ATC

Función:	PonerRetraso
Quien la llama:	- Comportamiento RecepcionAvion cuando llega un mensaje con OpcionMensaje=4
Datos de entrada:	- String Retraso
Actuación:	Le llega un retraso y lo guarda en la lista Retrasotiempo(tipo 1), Retrasoavion (tipo 2 y 3) o Retrasopista (tipo 4)
Funciones a las que llama:	- Tempsqueretrasem (CDM)
Datos Salida:	- ArrayList Retrasotiempo - ArrayList Retrasoavion - ArrayList Retrasopista

Función:	BuscarRetrasosTiempos
Quien la llama:	- Comportamiento RecepcionAvion cuando llega un mensaje con OpcionMensaje=5
Datos de entrada:	- String posicio - int encontrados - ArrayList Retrasotiempo
Actuación:	Encuentra todos los aviones afectados por un retraso de tipo 1, y escribe los retrasos cómo tipo 3
Funciones a las que llama:	- GetTempsPreviMil (CDM) - GetTempsPostMil (CDM) - restatemps (CDM) - MirarmodeliAO (CDM)

Datos Salida:	- String LlistaRetrasos - ArrayList Retrasotiempo

Función:	AplicarRetras
Quien la llama:	- Comportamiento RecepcionAvion cada vez que tiene de aplicar un retraso.
Datos de entrada:	- String[ ] Informació : información de un retraso tipo 3 - String hora
Actuación:	Aplica un retraso de tipo 3 a cualquier avión
Funciones a las que llama:	- TreureTemps (CDM) - TreureActualizada (CDM) - sumatemps (CDM) - PublicarInformacion (CDM)
Datos Salida:	-

Función:	AddSeparationTime
Quien la llama:	- Comportamiento RecepcionAvion cuando se le avisa de que es la hora de aplicar una reducción de la capacidad de pista. OpcionMensaje=7
Datos de entrada:	- String reduccio
Actuación:	- Calcula a que valor tenemos de reducir la capacidad de pista
Funciones a las que llama:	-
Datos Salida:	- boolean[ ] aplicar: 2 booleans. El primero indica si tenemos reducción de la capacidad en la pista de salida. El segundo si tenemos reducción de la capacidad en la pista de llegada. - ArrayList SeparationsTimeReducted

Función:	PosarSeparacionsReduccioCapacitat
Quien la llama:	- Comportamiento RecepcionAvion cuando OpcionMensaje=7
Datos de entrada:	- String reduccio - int arribada: 1=arribada, 0 = Salida - String HoraActual
Actuación:	- Assegurar que todos los aviones de llegada o salida mantengan las distancias de seguridad necesarias teniendo en cuenta las capacidades reducidas
Funciones a las que llama:	- GetSeparation - restatemps - PublicarInformacion (CDM) - GetSeparationTime
Datos Salida:	-

Función:	GetSeparationTime
Quien la llama:	- PosarSeparacioReduccioCapacitat - PistaOcupada

Datos de entrada:	- int A - int B - int Diferenciatemps - boolean Landing - ArrayList SeparationTimein - ArrayList SeparationTimeout
Actuación:	Busca la separación mínima necesaria entre 2 aviones
Funciones a las que llama:	-
Datos Salida:	- int Separation

Función:	GetSeparation
Quien la llama:	- PosarSeparacionsReduccioCapacitat - Pistaoocupada
Datos de entrada:	- int Hora - int arr: arr=1→ aterrizajes; arr=0→despegues - ArrayList SeparationsTimeReducted
Actuación:	Mira si a la hora tenemos una reducción de la capacidad de la pista, en caso afirmativo devuelve el mínimo tiempo necesario entre 2 aviones
Funciones a las que llama:	-
Datos Salida:	- int minutos

Función:	ActualizarSeparationTimes
Quien la llama:	- Comportamiento RecepcionAvion cuando llega un mensaje con OpcionMensaje=3
Datos de entrada:	- int ophINnominal - int ophOUTnominal
Actuación:	Varia los valores de separación entre aviones si el documento airport.txt marca unos valores de operaciones por hora inferior al nominal(30)
Funciones a las que llama:	-
Datos Salida:	- int[ ] SeparationTimeout - int[ ] SeparationTimein

Función:	Pistaocupada
Quien la llama:	- Comportamiento RecepcionAvion cuando se tiene de calcular ELDT o ETOT
Datos de entrada:	- int nuevoT - Aircraft avion - boolean landing - ArrayList LandingSequence - ArrayList TakeOffSequence - int politicacongestio
Actuación:	Mira si para un tiempo ELDT o ETOT la pista está ocupada, en caso afirmativo calcula el nuevo ETOT o ELDT del vuelo, o aplica retrasos a los aviones que sea necesario en función de la políticacongestio

Funciones a las que llama:	- AddSequence - GetPositionSequence - GetTimeSequence - restatems (CDM) - GetSeparationTime - GetSeparation - sumaTemps (CDM) - GetIdSequence - GetTempsActualMil (CDM) - PublicarInformacion (CDM) - mensajeMilestone (CDM)
Datos Salida:	- int nuevoT

Función:	getIdSequence
Quien la llama:	- Pistaocupada
Datos de entrada:	- int position - boolean landing - ArrayList LandingSequence - ArrayList TakeOffSequence
Actuación:	Devuelve el ID del avión que despega o aterriza en la position según la secuencia de takeoff y de landing
Funciones a las que llama:	-
Datos Salida:	- String ID

Función:	GetCategoriaSequence
Quien la llama:	- Pistaocupada
Datos de entrada:	- int position - boolean landing - ArrayList LandingSequence - ArrayList TakeOffSequence
Actuación:	Devuelve la categoría del avión que despega o aterriza en la position según la secuencia de takeoff y de landing
Funciones a las que llama:	-
Datos Salida:	- int cat

Función:	- GetTimeSequence
Quien la llama:	- Pistaocupada
Datos de entrada:	- int position - boolean landing - ArrayList LandingSequence - ArrayList TakeOffSequence
Actuación:	Devuelve el tiempo de despegue o aterrizaje del avión en la position según la secuencia de takeoff y de landing
Funciones a las que llama:	-
Datos Salida:	- int time

Función:	GetPositionSequence
Quien la llama:	- Pistaocupada
Datos de entrada:	- String Id - boolean landing - ArrayList LandingSequence - ArrayList TakeOffSequence
Actuación:	Le llega un Id de un avión y te devuelve un entero que indica en qué posición realiza el aterrizaje o el despegue.
Funciones a las que llama:	-
Datos Salida:	- int valor

Función:	AddSequence
Quien la llama:	- - Comportamiento RecepcionAvion - Pistaocupada
Datos de entrada:	- String Id - int tiempo - int categoria - boolean landing - ArrayList LandingSequence - ArrayList TakeOffSequence
Actuación:	Añade a la secuencia el avión con Id de entrada con su tiempo de salida o llegada. Si ese avión ya estaba dentro de la secuencia modifica su tiempo.
Funciones a las que llama:	-
Datos Salida:	- ArrayList LandingSequence - ArrayList TakeOffSequence

F.2.8 Métodos objeto Aircraft

Función:	AddActualizadas
Datos de entrada:	- int[ ] actu1 - int[ ] actu2
Actuación:	Crea un vector de actualizadas que contiene todos los valores de los 2 de entrada, si un valor aparece actualizado en ambas listas, el solo lo considera 1 vez.
Funciones a las que llama:	-
Datos Salida:	- int[ ] acutalizadas

Función:	añadirIncidencia
Datos de entrada:	- int tiempoAntiguo - int tiempoNuevo - String variable - String nombreAgente - int hora
Actuación:	Añade una incidencia a la lista de incidencias de un avión siempre que este cumpla con una serie

	de restricciones, que esa incidencia concreta no se ha añadido anteriormente, que el tiempo antiguo coincide con el que realmente era el tiempo antiguo, y que el tiempoNuevo no se ha adelantado respecto el tiempo real del avión estudiado.
Funciones a las que llama:	- getIncidencias - setIncidencias
Datos Salida:	-

Función:	añadirIncidenciaLineaCompleta
Datos de entrada:	- String linea - String Incidencias
Actuación:	Añade una incidencia a la lista de incidencias de un avión sin realizar ninguna comprobación.
Funciones a las que llama:	-
Datos Salida:	- String Incidencias

Función:	newStatus
Datos de entrada:	- int opcion: el valor del entero corresponde al índice del nuevo Status según el siguiente vector: [INITIATED, AIRBONE, FIR, FINAL, LANDED, INBLOCK, SEQUENCED, BOARDING, READY, OFFBLOCK, DEPARTED]
Actuación:	Varia el Status de un avión
Funciones a las que llama:	-
Datos Salida:	-

Función:	printFlight
Datos de entrada:	- int opcion
Actuación:	Muestra por pantalla la información relacionada con el avión. Según el número de opcion se mostrará más o menos información.
Funciones a las que llama:	-
Datos Salida:	-

F.2.9 Métodos objeto Tabla

Función:	printTabla
Datos de entrada:	- int opcion
Actuación:	Pinta todos los aviones de la tabla en pantalla. Según el valor de opcion mostrará más o menos información de cada avión.
Funciones a las que llama:	- printFlight (Aircraft)
Datos Salida:	-

Función:	PosicionTabla
Datos de entrada:	- Aircraft avion

Actuación:	Busca un avión en la tabla y devuelve su posición, si ese avión no aparece en la tabla lo indica y devuelve la posición que ocuparía si se añadiese en ese momento a la tabla.
Funciones a las que llama:	-
Datos Salida:	- Int[2] posicion: Primer valor indica la posición del avión en la tabla o la posición que tendría si se añadiera a la tabla. El segundo indica si ese avión ya está en la tabla

Función:	GetPositionAvion
Datos de entrada:	- String id
Actuación:	Busca un id en la tabla, si lo encuentra devuelve el indice de su posición en la tabla.
Funciones a las que llama:	-
Datos Salida:	- int posicion

Función:	contarAviones
Datos de entrada:	-
Actuación:	Cuenta la cantidad de aviones existentes en la tabla y devuelve el número, a la vez que actualiza el parámetro número de la tabla
Funciones a las que llama:	-
Datos Salida:	- int num

F.2.10 Métodos MilestoneTrigger

Función:	Variaciones
Quien la llama:	- Comportamiento OrdenarMilestones al recibir un mensaje avisando de nuevos tiempos de milestone
Datos de entrada:	- String[ ] text
Actuación:	Escribe un String[ ] a formato ArrayList
Funciones a las que llama:	-
Datos Salida:	- ArrayList

Función:	Variante
Quien la llama:	- Comportamiento OrdenarMilestones al recibir un mensaje avisando de nuevos tiempos de milestone
Datos de entrada:	- ArrayList Arr - String text
Actuación:	Añade un milestone a la lista de milestones (Arr). Si este existía y solo se le ha cambiado el tiempo al milestone, se borra el antiguo de la lista
Funciones a las que llama:	-
Datos Salida:	- ArrayList Arr

Función:	cargarCDM
Quien la llama:	- Comportamiento IniciarSimulacion al recibir un mensaje con OpcionMensake = 1
Datos de entrada:	- String ruta
Actuación:	Llama a la función que leerá el documento de TriggerCDM.txt
Funciones a las que llama:	- leerArchivoCDM
Datos Salida:	- ArrayList CDM

Función:	leerArchivoCDM
Quien la llama:	- cargarCDM
Datos de entrada:	- String ubicacion
Actuación:	función que leerá el documento de TriggerCDM.txt
Funciones a las que llama:	-
Datos Salida:	- ArrayList arrlist

Función:	SetMensaje
Quien la llama:	- Comportamiento OrdenarMilestones al recibir un mensaje con OpcionMensaje=1
Datos de entrada:	- ACLMessage mensaje
Actuación:	Copia el mensaje de entrada en MensajeInicio, para que el comportamiento IniciarSimulacion pueda leerlo
Funciones a las que llama:	-
Datos Salida:	- ACLMessage MensajeInicio

Función:	GetMensaje
Quien la llama:	Comportamiento IniciarSimulacion
Datos de entrada:	- ACLMessage MensajeInicio
Actuación:	Copia el mensaje de MensajeInicio, para que el comportamiento IniciarSimulacion pueda usarlo
Funciones a las que llama:	-
Datos Salida:	- ACLMessage mensaje

Función:	setMilestoneslist
Quien la llama:	- Comportamiento OrdenarMilestones
Datos de entrada:	- ArrayList Milestones
Actuación:	Actualitza la llista de milestones
Funciones a las que llama:	-
Datos Salida:	- ArrayList Milestoneslist

Función:	getMilestones
Quien la llama:	- Comportamiento IniciarSimulacion, a cada tic del timer - Comportamiento OrdenarMilestones
Datos de entrada:	- ArrayList Milestoneslist
Actuación:	Solicita la lista de milestones
Funciones a las que llama:	-
Datos Salida:	- ArrayList Milestoneslist

Función:	TriggerMilestone
Quien la llama:	- Comportamiento IniciarSimulacion, a cada tic del timer, que es cuando quiere mandar la información de un milestone. A cada tic un milestone.
Datos de entrada:	- String[ ] trigger - int GHamount - String AOforGH - String aerolineas
Actuación:	Crea el avión con la información que quiere mandar en cada caso. En función del milestone y de la aerolínea del avión del que quiere avisar, los receptores del mensaje serán unos u otros. A continuación procederá a llamar a una función que se encargara de mandar el mensaje
Funciones a las que llama:	- enviarTrigger - AerolineaAvion
Datos Salida:	-

Función:	enviarTrigger
Quien la llama:	- TriggerMilestone
Datos de entrada:	- Aircraft avion - AID[ ] AgenteReceptor
Actuación:	Envía un mensaje con la información del avion a los receptores
Funciones a las que llama:	-
Datos Salida:	-

Función:	AerolineaAvion
Quien la llama:	- TriggerMilestone
Datos de entrada:	- Aircraft avion - String aerolineas
Actuación:	Busca el nombre del agente Aerolinea que recibirá el mensaje con la información del avion
Funciones a las que llama:	-
Datos Salida:	- AID Aerolinea

F.2.11 Métodos Simulator Input

Función:	ActualizarAOforGH
Quien la llama:	Comportamiento RecepcionAvion
Datos de entrada:	-StringAOforGH
Actuación:	Actualizamos en avionparametros la línea de AOforGH
Funciones a las que llama:	-
Datos Salida:	-Aircraft avionparametros

Función:	EscanejarAOforGH
Quien la llama:	Comportamiento RecepcionAvion
Datos de entrada:	-String AOforGH
Actuación:	Mira que el String AOforGH este bien escrito, y en caso de tener algún elemento repetido lo elimina.
Funciones a las que llama:	-CargarAgentesAO
Datos Salida:	-String AOforGH

Función:	CargaragentesAO
Quien la llama:	Función EscanejarAOforGH
Datos de entrada:	-String AOforGH
Actuación:	Escribimos la lista de receptores de los mensajes con las empresas GH con agente propio
Funciones a las que llama:	-
Datos Salida:	-AID[ ] Agentes

Función:	Cargararchivos
Quien la llama:	Comportamiento RecepcionAvion
Datos de entrada:	-Aircraft a
Actuación:	Llama a la función que leerá los documentos input y escribirá los documentos intermedios. Comprueba que todo sale bien.
Funciones a las que llama:	-cargarAerolineas -leerInput -ArchivoAO
Datos Salida:	-

Función:	ArchivoAO
Quien la llama:	-funcion Cargararchivos
Datos de entrada:	-String rutaAO1 -String rutaAO2 -String rutaAO
Actuación:	Lectura de los documentos AO1 y AO2 para escribir el documento AO (vuelos aerolínea).
Funciones a las que llama:	-
Datos Salida:	-documento vuelos aerolínea -int 0/1: 0 si se ha escrito el documento sin problemas. 1 si el programa no ha podido escribir el documento.
Función:	CargarAerolineas
Quien la llama:	-funcion Cargararchivos
Datos de entrada:	-Aircraft avion
Actuación:	Busca en los parámetros la lista de aerolíneas con agente propio
Funciones a las que llama:	-
Datos Salida:	-String[ ] Aerolineas

Función:	leerInput
Quien la llama:	-funcion Cargararchivos

Datos de entrada:	-String rutaInput -String rutaFP -String rutaAO1 -String rutaAO2 -String rutaFPS
Actuación:	Leerá los archivos y devolverá 1 o 0 en función de si ha podido escribir cada uno de los 4 documentos (AO1, AO2, FP, FPS)
Funciones a las que llama:	-valores
Datos Salida:	-int [4] resultado -documento FlightPlans -documento FlightPlansSalidas -documento AO1 -documento AO2

Función:	RetrasosYMilestones
Quien la llama:	Comportamiento RecepcionAvion
Datos de entrada:	-boolean retrasos -String rutaMileatones
Actuación:	Creación del documento TriggerCDM teniendo en cuenta los retrasos si tenemos.
Funciones a las que llama:	-OrdenarLlista -leerInputRetrasos -MandarRetrasos -valores
Datos Salida:	-documento TriggerCDM

Función:	InicioPrograma
Quien la llama:	Comportamiento RecepcionAvion
Datos de entrada:	-Aircraft avionparametros
Actuación:	Envía un mensaje a los agentes avisando de que empezamos la simulación
Funciones a las que llama:	-
Datos Salida:	-

Función:	valores
Quien la llama:	-funcion RetrasosYMilestones -funcion leerInput
Datos de entrada:	-int[ ] vector
Actuación:	Encuentra el valor mínimo de un vector y su posición en el vector
Funciones a las que llama:	-
Datos Salida:	-int[2] valores

Función:	avisarDDR2
Quien la llama:	Comportamiento RecepcionAvion
Datos de entrada:	-boolean DDR2leido -Aircraft Avion

Actuación:	Envía un mensaje al agente DDR2 para que lea el documento SO6
Funciones a las que llama:	-
Datos Salida:	-

Función:	valorparametro
Quien la llama:	-funcion rutaArchivos
Datos de entrada:	-String parametro
Actuación:	Devuelve el segundo valor de un string si separamos por \t
Funciones a las que llama:	-
Datos Salida:	-String valor

Función:	rutaArchivos
Quien la llama:	-funcion definirRutas
Datos de entrada:	-Aircraft avion
Actuación:	Devuelve la parte común de la ubicación de los documentos
Funciones a las que llama:	-valorparametro
Datos Salida:	-String ubicacion

Función:	definirRutas
Quien la llama:	Comportamiento RecepcionAvion
Datos de entrada:	-Aircraft avion
Actuación:	Devuelve la ubicación de todos los documentos que se leerán o escribirán
Funciones a las que llama:	-rutaArchivos
Datos Salida:	-String rutaInput -String rutaFP -String rutaFPS -String rutaMilestones -String rutaAO1 -String rutaAO2 -String rutaAO -String rutaRetrasos

Función:	BroadcastParametros
Quien la llama:	Comportamiento RecepcionAvion
Datos de entrada:	-Aircraft avion -String AOforGH
Actuación:	Envío del mensaje con los parámetros a los agentes
Funciones a las que llama:	-
Datos Salida:	-

Función:	listaAerolineas
Quien la llama:	Comportamiento RecepcionAvion
Datos de entrada:	-Aircraft avionparametros (de la interfaz)

Actuación:	Utilizamos el avionparametros para obtener la lista de aerolineas
Funciones a las que llama:	-
Datos Salida:	-AID[ ] AgentesAO

Función:	leerInputretrasos
Quien la llama:	-funcion RetrasosYMilestones
Datos de entrada:	-String rutaRetrasos
Actuación:	Lectura del documento retrasos
Funciones a las que llama:	-
Datos Salida:	-String[ ] Retrasos

Función:	MandarRetrasos
Quien la llama:	-funcion RetrasosYMilestones
Datos de entrada:	-int numRetrasos -String[ ] Retrasos
Actuación:	Selección de uno de los retrasos y llamamos a la función que mandará la información sobre este. Se mandará un mensaje por cada retraso.
Funciones a las que llama:	-BroadcastInfoRetraso
Datos Salida:	-int r: r=1 Retrasos enviados sin problemas, r=0 algún retraso no se ha podido mandar correctamente.

Función:	BroadcastInfoRetraso
Quien la llama:	-funcion MandarRetrasos
Datos de entrada:	-Aircraft avion
Actuación:	Recibimos un retraso, y buscamos el agente interesado en recibir esta información y se la mandamos.
Funciones a las que llama:	-GetReceptor
Datos Salida:	-

Función:	GetReceptor
Quien la llama:	-funcion BroadcastInfoRetraso
Datos de entrada:	-Aircraft avion -ArrayList TiemposRetrasoMilestone -String AOforGH
Actuación:	Buscamos el agente interesado en recibir la información sobre el retraso que informaremos.
Funciones a las que llama:	-GetGHAID -GetAOAID
Datos Salida:	-AID[ ] receptor -boolean errorretrasoactual

Función:	GetGHAID
Quien la llama:	- función GetReceptor
Datos de entrada:	-String ID -String AOforGH

Actuación:	Buscamos el agente GH que tiene el contrato de un avión en concreto
Funciones a las que llama:	-
Datos Salida:	-AID GHAID

Función:	GetAOAID
Quien la llama:	- función GetReceptor
Datos de entrada:	-String ID -String[ ] Aerolineas
Actuación:	Buscamos el agente AO al que pertenece un avión, si este no tiene agente propio lo asignamos en el General.
Funciones a las que llama:	-
Datos Salida:	-AID Aerolinea

Función:	OrdenarLlista
Quien la llama:	-funcion RetrasosYMilestones
Datos de entrada:	-ArrayList Lista
Actuación:	Ordenamos la lista de Milestones y retrasos según la hora a la que sucede cada acción.
Funciones a las que llama:	-
Datos Salida:	-ArrayList Out

Función:	LlegirResults
Quien la llama:	Comportamiento RecepcionAvion
Datos de entrada:	-String Localitzacio
Actuación:	Envía un mensaje al Output con la ruta para abrir el documento de Output
Funciones a las que llama:	-
Datos Salida:	-

Función:	getAirport
Quien la llama:	Comportamiento RecepcionAvion
Datos de entrada:	-String ruta
Actuación:	Abre el documento que se encuentra en la ruta de entrada, y devuelve la primera línea del documento que contendrá el nombre del aeropuerto de estudio.
Funciones a las que llama:	-
Datos Salida:	-String Airport

F.2.12 Métodos Simulador Gráfica

Función:	PosicionTabla
Quien la llama:	- Comportamiento RecepcionAvion cuando llega un mensaje
Datos de entrada:	- Aircraft Avion

Actuación:	Busca si el avión ya se ha definido en la tabla. Devuelve la posición que ocupa.
Funciones a las que llama:	-
Datos Salida:	- Int[ ] posición. Si el avión no se encuentra en la tabla devuelve un posición = -1.

Función:	PararSimulación
Quien la llama:	- Interfaz (Botón “Parar Simulación”)
Datos de entrada:	-
Actuación:	Genera y envía un mensaje para avisar a los agentes de que hay de parar que la simulación. Define Output, InformationSharing y Milestone Trigger cómo receptores y añade las características del mensaje que finaliza la simulación.
Funciones a las que llama:	- enviarTrigger()
Datos Salida:	

Función:	enviarTrigger
Quien la llama:	- función PararSimulacion
Datos de entrada:	- Aircraft avion - AID[ ] AgenteReceptor
Actuación:	Envía un mensaje con la información del avión a los receptores
Funciones a las que llama:	-
Datos Salida:	-

F.2.13 Métodos Output

Función:	EscribirText
Quien la llama:	Comportamiento AvionenTabla al recibir un mensaje con OpcionMensaje=10
Datos de entrada:	- String ruta
Actuación:	Escribe en el documento que se encuentra en ruta la información relacionada con los aviones guardados en tabla
Funciones a las que llama:	-
Datos Salida:	- tabla tabla

F.2.14 Métodos DDR2

Función:	leerSO6
Quien la llama:	- Comportamiento SO6in
Datos de entrada:	- String rutaSO6: ruta para encontrar el fichero SO6 y poder leerlo. - String Airport: código ICAO del aeropuerto a estudiar

	- String[ ] WPFIR: Conjunto de puntos que forman la FIR del aeropuerto seleccionado String[ ]: WPAPP: Conjunto de puntos desde donde se empieza la aproximación al aeropuerto seleccionado
Actuación:	Esta función realiza casi todo el trabajo del agente DDR2, es la encargada de procesar el documento de entrada DDR2 para sacar por escrito el documento Input.txt
Funciones a las que llama:	- Calculotiempo - RelacionarAvions - MirarWP - GetAO
Datos Salida:	- docuemnto Input.txt

Función:	CalculoTiempo
Quien la llama:	- leerSO6 - RelacionarAvions
Datos de entrada:	Int tiempo Int delay: valor a sumar o restar al tiempo Int operación: 1→sumar 2→restar
Actuación:	Suma o resta los dos valores de entrada. Tiene en cuenta posibles cambios de hora
Funciones a las que llama:	-
Datos Salida:	- int result: resultado final

Función:	RelacionarAvions
Quien la llama:	leerSO6
Datos de entrada:	ArrayList Out: Lista con la información reducida de todos los datos de salidas de nuestro aeropuerto ArrayList In: Lista con la información reducida de todos los datos de llegadas a nuestro aeropuerto ArrayList Salidas: Lista con la información detallada de todos los datos de llegadas a nuestro aeropuerto ArrayList Llegadas: Lista con la información detallada de todos los datos de llegadas a nuestro aeropuerto
Actuación:	Encuentra los casos en los que es posible que el avión que realiza una salida sea el mismo que ha llegado al aeropuerto anteriormente.
Funciones a las que llama:	- OrdenarLlista
Datos Salida:	ArrayList Salidas: Lista con la información detallada de todos los datos de llegadas a nuestro aeropuerto, ahora contiene también las relaciones con los aviones de entrada

Función:	OrdenarLlista
Quien la llama:	- RelacionarAvions
Datos de entrada:	- ArrayList Llista
Actuación:	Ordena temporalmente todos los elementos de la lista.
Funciones a las que llama:	-
Datos Salida:	- ArrayList Out: Es la ArrayList de entrada ordenada

Función:	MirarWP
Quien la llama:	- LeerSO6
Datos de entrada:	- String[ ] WaypointsZona - String SegmentWP
Actuación:	Mira si el WayPoint de inicio de un segmento de vuelo corresponde con alguno de los de la lista de entrada (WPFIR o WPAPP)
Funciones a las que llama:	-
Datos Salida:	Boolean encontrado: false → No coincide true→ Si coincide

Función:	GetAO
Quien la llama:	- LeerSO6
Datos de entrada:	- String ID
Actuación:	A partir de la ID de un vuelo encuentra la aerolínea del avión, 3 primeras letras del ID
Funciones a las que llama:	-
Datos Salida:	String AO

Función:	MandarInfo
Quien la llama:	- Comportamiento SO6in después de leer el documento
Datos de entrada:	- boolean leido
Actuación:	Completa el avión que se mandara al SimulatorInput para informarle de cómo a ido la lectura del documento
Funciones a las que llama:	- BroadcastInfo
Datos Salida:	-

Función:	BroadcastInfo
Quien la llama:	- MandarInfo
Datos de entrada:	- Aircraft avion
Actuación:	mandara un mensaje al SimulatorInput para informarle de cómo a ido la lectura del documento y si ya se ha escrito el documento input.txt
Funciones a las que llama:	-
Datos Salida:	-

F.2.15 Métodos Turn Around

Función:	LlegirText
Quien la llama:	- Comportamiento BuscarTurnAround al recibir el mensaje de inicialización.
Datos de entrada:	- String ruta: Ruta para encontrar el documento Turn- Around- Time.txt
Actuación:	Realiza una lectura del documento, y se guarda la información extraida en 2 ArrayList.
Funciones a las que llama:	-
Datos Salida:	- Boolean leído: Indica si se ha leído correctamente el documento. - ArrayList Modelos: Lista con todos los nombres de los modelos que aparecían en el documento. - ArrayList Tiempos: Lista con todos los tiempos de turn around para cada modelo de avión del documento.

Función:	GetTurnAroundTime
Quien la llama:	- Comportamiento BuscarTurnAround cada vez que recibe un mensaje preguntando el tiempo de un Turn Around
Datos de entrada:	- String Modelo: Modelo del avión del que queremos saber el turn around time. - Int cat: Categoría del avión del que queremos saber el turn around time. - boolean leído: Indicador de si se ha leído bien el documento Turn- Around- Time.txt
Actuación:	Es el encargado de organizar el proceso para obtener el turn around time del avión solicitado en cada caso.
Funciones a las que llama:	- BuscarTAT - TATcasgeneral
Datos Salida:	- Int TurnAroundTime: Valor del tiempo mínimo de turna round time del avión seleccionado.

Función:	BuscarTAT
Quien la llama:	- Funcion GetTurnAroundTime
Datos de entrada:	- String Modelo: Modelo del avión del que queremos saber el turn around time. - ArrayList Modelos: Lista con todos los nombres de los modelos que aparecían en el documento. - ArrayList Tiempos: Lista con todos los tiempos de turn around para cada modelo de avión del documento.
Actuación:	Busca el TAT para e la lista para el modelo de entrada. Si el modelo no aparece en la lista, mira si aparece en la ArrayList el caso GEN
Funciones a las que llama:	-

Datos Salida:	- Int[2] Info: Primer valor Turn Around Time, segundo valor indica cómo se ha obtenido el primero.

Función:	TATcasgeneral
Quien la llama:	- Funcion GetTurnAroundTime
Datos de entrada:	- Int[2] Info: Primer valor Turn Around Time, segundo valor indica cómo se ha obtenido el primero. - Int cat: Categoría del avión estudiado.
Actuación:	Añade una constante de tiempo al TurnAroundTime en función de la categoría del avión. Solo si el documento Turn- Around-Times.txt no contiene información de nuestro modelo en concreto.
Funciones a las que llama:	-
Datos Salida:	- Int TurnAroundTime

Función:	GetReceptor
Quien la llama:	- Comportamiento BuscarTurnAround al recibir el mensaje de inicialización, para saber a quién responder.
Datos de entrada:	- String AO: Aerolinea del avión - int GHamount: cantidad de empresas de GH existentes en nuestro simulador - String AOforGH: Indicador de los contratos de los diferentes GH.
Actuación:	Sabiendo los contratos de las empresas de GH y la aerolínea del avión que estamos estudiando, buscará cual es la empresa a la que tiene de mandar la información con el turn around time.
Funciones a las que llama:	-
Datos Salida:	- AID GHAID

Función:	MandarInfoGH
Quien la llama:	-Comportamiento BuscarTurnAround, para mandar la respuesta con el turn around time
Datos de entrada:	- Aircraft avión: Siempre se manda la información en una clase avión. En las incidencias tendrá el turn around time. - int Turna round time - AID Receptor: es el mismo AID encontrado en la función GetReceptor.
Actuación:	Compila el mensaje con los json y lo manda al GH.
Funciones a las que llama:	-
Datos Salida:	-

F.3 Métodos Interfaces

F.3.1 Interfaz

Función:	datosAvion
Quien la llama:	- ActualizarTabla
Datos de entrada:	- Aircraft avion
Actuación:	Traduce el objeto Aircraft a un String[ ], con el que trabajará la interfaz.
Funciones a las que llama:	- ArreglarMuestra
Datos Salida:	- String[ ] datos

Función:	ActualizarTabla
Datos de entrada:	- Aircraft avion - Int posicion
Actuación:	Añade un avión a la tabla que se muestra al usuario. Comprueba si el avión está o no en la tabla para actualizarlo o añadirlo al final.
Funciones a las que llama:	- DatosAvion - AircraftsNumber - ActualizarValores
Datos Salida:	-

Función:	AircraftNumber
Quien la llama:	- función ActualizarTabla
Datos de entrada:	- Tabla table
Actuación:	Cuenta el número de aviones en la tabla, diferenciando si están AIRBORNE o ON AIRPORT
Funciones a las que llama:	-
Datos Salida:	- int[ ] Aircrafts. Devuelve 3 valores. El número total de aviones en el CDM, el número de aviones AIRBORNE y el número de aviones ON AIRPORT

Función:	ActualizarValores
Quien la llama:	- ActualizarTabla
Datos de entrada:	- Aircraft Avion - Objeto[ ] datos
Actuación:	Busca en que posición de la tabla se encuentra el avión y lo substituye con los nuevos datos
Funciones a las que llama:	- Posicionenlatabla
Datos Salida:	-

Función:	Posicionenlatabla
Quien la llama:	- ActualizarValores
Datos de entrada:	- Object[ ] datos
Actuación:	Busca el avión en la tabla y devuelve la posición en la que se encuentra. Para la búsqueda comprueba el ID del avión.

Funciones a las que llama:	-
Datos Salida:	- int numero. La posición del avión en la tabla

Función:	ArreglarMuestra
Quien la llama:	-funcion datosAvion
Datos de entrada:	-String t
Actuación:	Cambia el tiempo para dejarlo adecuado a cómo se verá en la tabla. Se busca que todos los tiempos tengan 4 cifras.
Funciones a las que llama:	-
Datos Salida:	-String newt. Tiempo arreglado para la visualización del usuario.

F.3.2 Interfaz SimulatorInput

Función:	definirParametros
Quien la llama:	-funcion CargarParametrosInicio
Datos de entrada:	- Valores de diferentes textbox de la interfaz
Actuación:	Creación del String de parámetros de entrada a partir de la interfaz
Funciones a las que llama:	-
Datos Salida:	- String txt

Función:	CargarArchivoResults
Quien la llama:	-Al pulsar el botón Cargar Resultados
Datos de entrada:	- String ruta
Actuación:	Comprueba que existe el archivo con los resultados y guarda la ruta cómo válida.
Funciones a las que llama:	-
Datos Salida:	- Boolean Valido - Aviso (JOptionPane) en caso de Valido = false.

Función:	CargarArchivoSO6
Quien la llama:	-Al pulsar el botón Cargar
Datos de entrada:	-String ruta -String So6
Actuación:	Comprueba que existe el archivo con la información DDR2
Funciones a las que llama:	-
Datos Salida:	- Boolean Valido - Aviso (JOptionPane) en caso de Valido = false.

Función:	CargarArchivoAirport
Quien la llama:	-Al pulsar el botón Cargar -funcion CargarParametrosInicio
Datos de entrada:	-String ruta
Actuación:	Comprueba que existe el archivo aeropuerto y lee la información que contiene

Funciones a las que llama:	-
Datos Salida:	- String parametros - Aviso (JOptionPane) en caso de error al leer el documento

Función:	guardarArchivo
Quien la llama:	-funcion CargarParametrosInicio
Datos de entrada:	-String ruta -String parametros
Actuación:	Escribe el documento parametros.txt
Funciones a las que llama:	-
Datos Salida:	-int : o→ Documento escrito correctamente -1→ No se ha podido escribir el documento -documento parámetros.txt

Función:	valorparametro
Quien la llama:	-funcion rutaArchivos -botón administrar retrasos
Datos de entrada:	-String parametro
Actuación:	Recibe un String con una serie de parámetros, y devuelve un String con el parámetro de la posición [1]
Funciones a las que llama:	-
Datos Salida:	-String valor

Función:	AvionParametro
Quien la llama:	-botón Iniciar Simulación -funcion CargarParametrosInicio
Datos de entrada:	-String parametros
Actuación:	Prepara el objeto Aircraft que se mandará con los parámetros
Funciones a las que llama:	-
Datos Salida:	-Aircraft avionparametros

Función:	mensajeAviso
Quien la llama:	Función avionesInput
Datos de entrada:	String aviso
Actuación:	Muestra un JOptionPane
Funciones a las que llama:	-
Datos Salida:	- Aviso (JOptionPane) en caso de error al leer el documento

Función:	avionesInput
Quien la llama:	-funcion CargarParametrosInicio
Datos de entrada:	String ruta
Actuación:	Cuenta la cantidad de aviones que se simularan (cuenta los aviones del documento input.txt)
Funciones a las que llama:	-mensajeAviso
Datos Salida:	-int count

Función:	CargarParametrosInicio
Quien la llama:	-Agente SimulatorInput
Datos de entrada:	-
Actuación:	Prepara todos los parámetros de inicio
Funciones a las que llama:	-definirParametros -AvionParametros -AvionesInput -CargarArchivoAirport -guardarArchivo -mostrarTiempo
Datos Salida:	-boolean guardado -boolean parametrosDefinidos

Función:	rutatextbox
Quien la llama:	-botón Cargar
Datos de entrada:	-Información de los TextBox de la interfaz
Actuación:	Escribe la Ruta según la información de la interfaz
Funciones a las que llama:	-
Datos Salida:	-String RUTA

Función:	mostrarTiempo
Quien la llama:	-funcionCargarParametrosIicio
Datos de entrada:	-int tiempo
Actuación:	Escribe el tiempo en hh/mm/ss, input en milisegundos
Funciones a las que llama:	-
Datos Salida:	-String salida

Función:	MostrarErrorDDR2
Quien la llama:	-Agente SimulatorInput
Datos de entrada:	-
Actuación:	Muestra por consola un error al leer el documento DDR2
Funciones a las que llama:	-
Datos Salida:	-

Función:	MostrarErrorDDR2noaviones
Quien la llama:	-Agente SimulatorInput
Datos de entrada:	-
Actuación:	Muestra por consola un error al leer el documento DDR2 porque no encuentra ningún avión que cumple los parámetros pedidos.
Funciones a las que llama:	-
Datos Salida:	-

Función:	BloquearParametros
Quien la llama:	-Agente SimulatorInput
Datos de entrada:	-

Actuación:	Bloquea los textBox para definir los parámetros, de manera que ya no puedan ser modificados
Funciones a las que llama:	-
Datos Salida:	-

F.3.3 Interfaz Retrasos

Función:	pasarALinea
Quien la llama:	-funcion actualizarTablas
Datos de entrada:	-String[ ] re
Actuación:	Convierte la información del retraso en tipo Object.
Funciones a las que llama:	-
Datos Salida:	-Object[ ]

Función:	actualizarTablas
Quien la llama:	-botón aceptar de InterfazFormularioRetrasos1 -funcion PintarTablas
Datos de entrada:	-String[ ] re
Actuación:	Decide donde guardar la información de cada uno de los retrasos.
Funciones a las que llama:	- pasarALinea
Datos Salida:	-DefaultTableModel modelo1 -DefaultTableModel modelo2 -DefaultTableModel modelo3 -DefaultTableModel modelo4

Función:	cargarTXT
Quien la llama:	-botón Cargar Retrasos
Datos de entrada:	-String ruta -ArrayList ListaRetrasos
Actuación:	Lee el documento de retrasos y coloca los retrasos leidos en cada tabla según correspondan
Funciones a las que llama:	- adaptarRetrasoLista -PintarTablas
Datos Salida:	-boolean correcto -Aviso (JOptionPane) en caso de error al leer el documento

Función:	PintarTablas
Quien la llama:	-funcion cargarTXT
Datos de entrada:	-ArrayList lista
Actuación:	Pinta todos los retrasos en las tablas
Funciones a las que llama:	-actualizarTablas
Datos Salida:	-

Función:	adaptarRetrasoLista
Quien la llama:	-funcion cargarTXT
Datos de entrada:	-String[ ] re

Actuación:	Adaptamos cualquier retraso a una única manera de definirlos
Funciones a las que llama:	-generarVec
Datos Salida:	-String[ ] retraso

Función:	generarVec
Quien la llama:	-funcion adaptarRetrasoLista
Datos de entrada:	-int l
Actuación:	Genera un vector de Strings con "-" en todas las posiciones
Funciones a las que llama:	-
Datos Salida:	-String[ ] r

Función:	GuardarRetrasosTXT
Quien la llama:	-botón aceptar -botón guardar (interfaz GuardarRetrasos)
Datos de entrada:	-String ruta
Actuación:	Escribe el fichero con la información de los retrasos
Funciones a las que llama:	-pasarAString
Datos Salida:	-documento retrasos

Función:	pasarAString
Quien la llama:	-funcion GuardarRetrasosTXT
Datos de entrada:	-String[ ] vec
Actuación:	Transforma la información de un retraso a formato String
Funciones a las que llama:	-
Datos Salida:	-String s

Función:	BorrarLinea
Quien la llama:	-botón borrar
Datos de entrada:	-int tabla -int linea
Actuación:	Elimina el retraso de línea en la tabla seleccionada
Funciones a las que llama:	-
Datos Salida:	-String s

F.3.4 Interfaz FormularioRetrasos

Función:	activarPaneles
Quien la llama:	-Al escoger algún parámetro en el ComboBox que selecciona el tipo de retraso.
Datos de entrada:	-int num -int numAviones
Actuación:	Escojemos que panel para escoger el retraso que mostraremos.
Funciones a las que llama:	-AerolineasModelosAviones

	-CargarComboBoxes -nombreSituacion
Datos Salida:	-

Función:	lecturaArchivo
Quien la llama:	-Al escoger algún parámetro en el ComboBox que selecciona el tipo de retraso
Datos de entrada:	-String RUTA
Actuación:	Leemos el documento vuelos Aerolinea, para tener la información sobre los aviones que podemos retrasar
Funciones a las que llama:	-contarAvionesArchivo -cargarDatos -AerolineasModelosAviones
Datos Salida:	-int numAviones -String[ ][ ] datos

Función:	contarAvionesArchivo
Quien la llama:	-función lecturaArchivo
Datos de entrada:	-String ruta
Actuación:	Leemos el documento con los aviones, para tener la información sobre los aviones que podemos retrasar
Funciones a las que llama:	-
Datos Salida:	-int count

Función:	cargarDatos
Quien la llama:	-función lecturaArchivo
Datos de entrada:	-String ruta
Actuación:	Leemos el documento con los aviones i guardamos toda la información
Funciones a las que llama:	-
Datos Salida:	-String[ ][ ] datos

Función:	AerolineasModelosAviones
Quien la llama:	-función activarPaneles -función lecturaArchivo
Datos de entrada:	-int numero String[ ][ ] datos
Actuación:	Saca los datos de aerolíneas y modelos
Funciones a las que llama:	-
Datos Salida:	-ArrayList aerolineas -ArrayList modelos -ArrayList aviones

Función:	CargarComboBoxes
Quien la llama:	-función activarPaneles
Datos de entrada:	-ArrayList modelos -ArrayList aviones

	-ArrayList aerolineas
Actuación:	Rellenamos los ComboBoxes con las opciones posibles
Funciones a las que llama:	-
Datos Salida:	-

Función:	CargarAvionesFiltrado
Quien la llama:	-Interfaz cada vez que se selecciona algún elemento en los ComboBox para definir el filtro
Datos de entrada:	-String aerolinea -String modelo -String[ ][ ] datos
Actuación:	Carga la lista de aviones que cumplen los requisitos del filtrado
Funciones a las que llama:	-
Datos Salida:	-ArrayList newAviones

Función:	actualizarComboBox
Quien la llama:	-Interfaz cada vez que se selecciona algún elemento en los ComboBox para definir el filtro
Datos de entrada:	-ArrayList lista -JComboBox combo
Actuación:	Actualiza la lista del ComboBox
Funciones a las que llama:	-
Datos Salida:	-

Función:	nombreSituacion
Quien la llama:	-funcion activarPaneles
Datos de entrada:	-int opcion
Actuación:	Devuelve una definición del motivo del retraso en función de opción (posición escojida del comboBox)
Funciones a las que llama:	-
Datos Salida:	-String nombre

Función:	getX
Quien la llama:	-funcion activarPaneles
Datos de entrada:	-String situacion
Actuación:	Devuelve el parámetro “X” a partir del nombre que tiene la situación
Funciones a las que llama:	-definirRetrasoTipo2 -definirRetrasoTipo1 -comprobarMilestoneCoherente
Datos Salida:	-String con el parámetro X

Función:	definirRetrasoVistaPrevia
Quien la llama:	-función definirRetrasoTipo1 -función definirRetrasoTipo2 -función definirRetrasoTipo3

	-función definirRetrasoTipo4
Datos de entrada:	-String[ ] re
Actuación:	Devuelve cualquier tipo de retraso en un String[13].
Funciones a las que llama:	-generarVec -getSituacion
Datos Salida:	-String[ ] retraso

Función:	generarVec
Quien la llama:	-función definirRetrasoVistaPrevia -función setRetrasoDef
Datos de entrada:	-int l
Actuación:	Genera un vector de Strings con "-" en todas las posiciones
Funciones a las que llama:	-
Datos Salida:	-String[ ] r

Función:	getSituacion
Quien la llama:	-funcion definirRetrasoVistaPrevia
Datos de entrada:	-int opcion
Actuación:	Devuelve una definición del motivo del retraso en función de opción (valor de *Situacion*)
Funciones a las que llama:	-
Datos Salida:	-String nombre

Función:	getdatos
Quien la llama:	-ComboBox de seleccionar avión
Datos de entrada:	-String ID
Actuación:	Devuelve los datos de un avión concreto
Funciones a las que llama:	-
Datos Salida:	-String[ ] d

Función:	actualizarTablaAvionSelect
Quien la llama:	- ComboBox de seleccionar avión
Datos de entrada:	-String[ ] dd -int tipo
Actuación:	Actualiza la tabla del avión seleccionado
Funciones a las que llama:	-
Datos Salida:	-

Función:	definirRetrasoTipo2
Quien la llama:	- Botón Vista Previa al definir retrasos tipo 2
Datos de entrada:	-valores de los TextBox y ComboBox de la interfaz
Actuación:	Guarda el retraso definido y lo muestra en el apartado de vista previa
Funciones a las que llama:	-comprobarMilestoneCoherente -sacarMilestone -setRetrasoDef -definirRetrasoVistaPrevia

	-VistaPreviaRetraso
Datos Salida:	-boolean definido

Función:	definirRetrasoTipo1
Quien la llama:	- Botón Vista Previa al definir retrasos tipo 1
Datos de entrada:	-valores de los TextBox y ComboBox de la interfaz
Actuación:	Guarda el retraso definido y lo muestra en el apartado de vista previa
Funciones a las que llama:	-sacarTiempo -comprobarMilestoneCoherente -sacarMilestone -setRetrasoDef -definirRetrasoVistaPrevia -VistaPreviaRetraso
Datos Salida:	-boolean definido

Función:	definirRetrasoTipo4
Quien la llama:	- Botón Vista Previa al definir retrasos tipo 4
Datos de entrada:	-valores de los TextBox y ComboBox de la interfaz
Actuación:	Guarda el retraso definido y lo muestra en el apartado de vista previa
Funciones a las que llama:	-sacarTiempo -comprobarMilestoneCoherente -setRetrasoDef -definirRetrasoVistaPrevia -VistaPreviaRetraso
Datos Salida:	-boolean definido

Función:	definirRetrasoTipo3
Quien la llama:	- Botón Vista Previa al definir retrasos tipo 3
Datos de entrada:	-valores de los TextBox y ComboBox de la interfaz
Actuación:	Guarda el retraso definido y lo muestra en el apartado de vista previa
Funciones a las que llama:	-comprobarMilestoneCoherente -sacarMilestone -setRetrasoDef -definirRetrasoVistaPrevia -VistaPreviaRetraso
Datos Salida:	-boolean definido

Función:	sacarMilestone
Quien la llama:	-función definirRetrasoTipo1 -función definirRetrasoTipo2 -función definirRetrasoTipo3 -función comprobarMilestoneCoherente -función sacarMilestoneParametro
Datos de entrada:	-String m
Actuación:	Sacamos únicamente el número de milestone de un string del ComboBox de milestones

Funciones a las que llama:	-
Datos Salida:	-String mm

Función:	VistaPreviaRetraso
Quien la llama:	-función definirRetrasoTipo1 -función definirRetrasoTipo2 -función definirRetrasoTipo3 -función definirRetrasoTipo4
Datos de entrada:	-String[ ] re
Actuación:	Asigna los valores de los parámetros del retraso a los labels de vista previa
Funciones a las que llama:	-
Datos Salida:	-

Función:	setRestrasosDef
Quien la llama:	-función definirRetrasoTipo1 -función definirRetrasoTipo2 -función definirRetrasoTipo3 -función definirRetrasoTipo4
Datos de entrada:	-String[ ] re
Actuación:	Definimos un String[ ] que es el que se mandará a la interfaz para informarle del retraso que se ha definido
Funciones a las que llama:	-
Datos Salida:	-String[ ] RetrasoDef

Función:	sacarTiempo
Quien la llama:	-función definirRetrasoTipo1 -función definirRetrasoTipo4
Datos de entrada:	-int t
Actuación:	Sacamos el tiempo en 2 cifras si este es menos a 10, de manera que 3h→03
Funciones a las que llama:	-
Datos Salida:	-String valor

Función:	resetVistaPrevia
Quien la llama:	-Selección de un tipo de retraso distinto al que se estaba definiendo anteriormente
Datos de entrada:	-
Actuación:	Las variables de vista previa vuelven a 0
Funciones a las que llama:	-
Datos Salida:	-

Función:	comprobarHorasCoherentes
Quien la llama:	-función definirRetrasoTipo1 -función definirRetrasoTipo4
Datos de entrada:	-String con -String inicio -String fin

Actuación:	Comprueba que se cumpla la relación Hcon<Hinicio<Hfinal
Funciones a las que llama:	-restatemps
Datos Salida:	-boolean coherente

Función:	restatemps
Quien la llama:	- función comprobarHorasCoherentes
Datos de entrada:	- int A - int B
Actuación:	A menos B, teniendo en cuenta hhmm
Funciones a las que llama:	-
Datos Salida:	- int C: resultado de la resta

Función:	comprobarMilestoneCoherente
Quien la llama:	-función definirRetrasoTipo2 -función definirRetrasoTipo3
Datos de entrada:	-String situacion -String MilestoneCon -int opcion
Actuación:	Comprueba que no se intente retrasar un milestone que en el momento de conocimiento del retraso ya se haya realizado
Funciones a las que llama:	-sacarMilestoneParametro
Datos Salida:	-boolean coherente

Función:	sacarMilestoneParametro
Quien la llama:	-función comprobarMilestoneCoherente
Datos de entrada:	-String parametro
Actuación:	Obtiene el milestone que corresponde a cada tiempo.
Funciones a las que llama:	-
Datos Salida:	-int milestone

F.3.5 Interfaz GuardarRetrasos

Función:	setRuta
Quien la llama:	-Interfaz Retrasos al abrir esta interfaz
Datos de entrada:	-String r
Actuación:	Guarda la ruta para el documento retrasos y la muestra por panalla
Funciones a las que llama:	-
Datos Salida:	-String ruta

F.3.6 Interfaz Output

Función:	setTabla
Quien la llama:	-Agente Output

Datos de entrada:	-Tabla t
Actuación:	Recibe una tabla de aviones y la guarda en la tabla de la memoria del agente
Funciones a las que llama:	-ActualizarTabla -AircraftsNumber
Datos Salida:	-Tabla tablaAviones

Función:	datosAvion
Quien la llama:	-función ActualizarTablaCargada -función ActualizarTabla
Datos de entrada:	-Aircraft avion
Actuación:	Traduce el objeto aircraft a un String[ ] con el que trabaja la interfaz
Funciones a las que llama:	-arreglarMuestra
Datos Salida:	-String[ ] datos

Función:	ActualizarTabla
Quien la llama:	-función setTabla
Datos de entrada:	-Aircraft avion -int posicion
Actuación:	Añadimos el avión a la tabla en la posición indicada
Funciones a las que llama:	-datosAvion -AircraftsNumber -ActualizarValores
Datos Salida:	-DefaultTableModel model -DefaultTableModel model1

Función:	ActualizarTablaCargada
Quien la llama:	-función cargardeTXT
Datos de entrada:	-Aircraft avion -int posicion
Actuación:	Añadimos el avión a la tabla en la posición indicada
Funciones a las que llama:	-datosAvion -AircraftsNumber -ActualizarValoresCargada
Datos Salida:	-DefaultTableModel model2

Función:	ActualizarValores
Quien la llama:	-función ActualizarTabla
Datos de entrada:	-Aircraft avion -Object[ ] datos
Actuación:	El avión ya esta en la tabla y ha cambiado algún valor, lo actualizamos
Funciones a las que llama:	-posicionenlatabla
Datos Salida:	-DefaultTableModel model -DefaultTableModel model1

Función:	ActualizarValoresCargada
Quien la llama:	-función ActualizarTablaCargada
Datos de entrada:	-Aircraft avion -Object[ ] datos
Actuación:	El avión ya esta en la tabla y ha cambiado algún valor, lo actualizamos
Funciones a las que llama:	-posicionenlatabla
Datos Salida:	-DefaultTableModel model2

Función:	posicionenlatabla
Quien la llama:	-función ActualizarValoresCargada -función ActualizarValores
Datos de entrada:	-Object[ ] datos
Actuación:	Busca la posición donde tenemos de añadir un avión en la tabla
Funciones a las que llama:	-
Datos Salida:	-int i

Función:	AircraftsNumber
Quien la llama:	-función ActualizarTabla -función ActualizarTablaCargada -función setTabla
Datos de entrada:	-Tabla table
Actuación:	Busca en la tabla cuantos aviones están en AIRBONE y cuantos ON AIRPORT
Funciones a las que llama:	-
Datos Salida:	-int[ ] aviones

Función:	arreglarMuestra
Quien la llama:	-función datosAvion
Datos de entrada:	-String t
Actuación:	Assegura que los tiempos tengan 4 valores: 435→0435
Funciones a las que llama:	-
Datos Salida:	-String newt

Función:	cargardeTXT
Quien la llama:	-Botón Cargar de TXT
Datos de entrada:	-String ruta
Actuación:	Lee el documento que se encuentra en ruta y carga la tabla con los valores del archivo
Funciones a las que llama:	-leerTabla -ActualizarTablaCargada
Datos Salida:	-Tabla tablaCargada

Función:	leerTabla
Quien la llama:	-función cargardeTXT
Datos de entrada:	-String r

Actuación:	Lee el documento que se encuentra en ruta y carga la tabla con los valores del archivo
Funciones a las que llama:	-
Datos Salida:	-Tabla tablavacia

Función:	clearTablaCargada
Quien la llama:	-Botón Cargar de TXT
Datos de entrada:	-
Actuación:	Borra las líneas de la tablaCargada
Funciones a las que llama:	-
Datos Salida:	-DefaultTableModel model2

Función:	setRutaGen
Quien la llama:	-Agente Output
Datos de entrada:	String r
Actuación:	Comprueba que la ruta termine con .txt, pues viene distinto si estamos cargando unos resultados, o queremos cargar los resultados de la simulación actual
Funciones a las que llama:	-
Datos Salida:	String RutaPrograma

F.3.7 Interfaz Analisis

Función:	ResetValores
Quien la llama:	-Botón analizar
Datos de entrada:	-
Actuación:	Vuelve los valores a los iniciales, para poder realizar un nuevo análisis posteriormente
Funciones a las que llama:	-
Datos Salida:	-int starttime -int finishtime -int starttime2 -int finishtime2 -int[3] avion1maximo -int[3] avion2maximo

Función:	EscribirResultados
Quien la llama:	-Botón analizar
Datos de entrada:	-Parámetros calculados durante el análisis
Actuación:	Escribe los datos calculados durante el análisis en un documento
Funciones a las que llama:	-
Datos Salida:	-documento analisis

Función:	setTabla
Quien la llama:	-Agente InterfazOutput
Datos de entrada:	-Tabla t -int opcion

Actuación:	Añadir una tabla de aviones a la tabla interna de la interfaz
Funciones a las que llama:	-ModelosyAO -CargarComboBoces
Datos Salida:	-Tabla tabla1 -Tabla tabla2

Función:	ModelosyAO
Quien la llama:	- función setTabla
Datos de entrada:	-Tabla tabla -ArrayList aerolíneas -ArrayList modelos
Actuación:	Busca todos los modelos y aerolíneas de la tabla y genera la lista
Funciones a las que llama:	-
Datos Salida:	-ArrayList aerolíneas -ArrayList modelos

Función:	CargarComboBoxes
Quien la llama:	- función setTabla
Datos de entrada:	-ArrayList aerolineas -ArrayList modelos
Actuación:	De las listas de modelos y aerolíneas llena las opciones de los comboBoxes para los filtros
Funciones a las que llama:	-
Datos Salida:	-

Función:	añadirAlFiltro
Quien la llama:	- Al Seleccionar algún elemento ComboBox aerolíneas o comboBox modelos
Datos de entrada:	-String tipo -Parametros Interfaz
Actuación:	Añade una aerolínea o un modelo al filtro
Funciones a las que llama:	-
Datos Salida:	-

Función:	actualizarFiltro
Quien la llama:	- Bootón Clear filtro - CheckBox sin filtro - Al Seleccionar algún elemento ComboBox aerolíneas o comboBox modelos
Datos de entrada:	-Parametros interfaz
Actuación:	Guarda los parámetros del filtro en la memoria
Funciones a las que llama:	-
Datos Salida:	-String aerolineasFiltro -String modelosFiltro

Función:	sacarParametros
Quien la llama:	- Al seleccionar algún parámetro en el ComboBox

Datos de entrada:	-String combo
Actuación:	Obtiene el parámetro del análisis de manera que el programa lo entienda
Funciones a las que llama:	-
Datos Salida:	-String parametro

Función:	gettimessimulacion
Quien la llama:	-Botón analizar
Datos de entrada:	-String tiempos -Tabla tabla1 -boolean rutas2 -Tabla tabla2
Actuación:	Calculamos los tiempos de inicio y final del analisis
Funciones a las que llama:	-gettemps
Datos Salida:	-int starttime -int finishtime -intstarttime1 -intfinishtime1 -intstarttime2 -intfinishtime2

Función:	restatemps
Quien la llama:	- Cualquiera que desee restar dos tiempos
Datos de entrada:	- int A - int B
Actuación:	A menos B, teniendo en cuenta hhmm
Funciones a las que llama:	-
Datos Salida:	- int C: resultado de la resta

Función:	sumatemps
Quien la llama:	- Cualquiera que desee sumar dos tiempos
Datos de entrada:	- int A - int min : no es una hora, es cantidad de minutos a sumar
Actuación:	Suma los minutos min al valor A, teniendo en cuenta hhmm
Funciones a las que llama:	-
Datos Salida:	- int C: resultado de la suma

Función:	gettemps
Quien la llama:	- función gettimessimulacion -función EscanearMirandotiempo(String, int, int) -función EscanearMirandotiempo(String)
Datos de entrada:	- Aircraft avion - String tiempo
Actuación:	Devuelve el valor de “tiempo” del avión
Funciones a las que llama:	-
Datos Salida:	- int[ ] valor

Función:	calculosventana
Quien la llama:	-Botón analizar
Datos de entrada:	-String tiempos -boolean hayventana -boolean rutas2 -int finishtimeVentana -int starttimeVentana
Actuación:	Prepara los vectores para la gráfica de la ventana
Funciones a las que llama:	- restatemps -EscanearMirandotiempo
Datos Salida:	- int[ ] analisisTabla1ventana - int[ ] analisisTabla2ventana - int minutostotalessimulacionventana - double[ ] vectorXventana

Función:	EscanearMirandotiempo(String, int, int)
Quien la llama:	-función calculosventana
Datos de entrada:	-String tiempos -int inicio -int fin -Tabla tabla1 -Tabla tabla2
Actuación:	Análisis tablas en la ventana
Funciones a las que llama:	-gettemps -restatemps
Datos Salida:	-Todos los parámetros que analizamos

Función:	EscanearMirandotiempo(String)
Quien la llama:	-Botón analizar
Datos de entrada:	-String tiempos -Tabla tabla1 -Tabla tabla2
Actuación:	Análisis tablas en toda la simulación
Funciones a las que llama:	-gettemps -restatemps
Datos Salida:	-Todos los parámetros que analizamos

Función:	setRuta
Quien la llama:	-InterfazOutput
Datos de entrada:	-String r
Actuación:	Define la Ruta donde escribiremos los resultados encontrados
Funciones a las que llama:	-cortarRuta
Datos Salida:	-String Ruta

Función:	contarAvionesTablas
Quien la llama:	-Botón Clear del filtro -Al Seleccionar algún elemento ComboBox aerolíneas o comboBox modelos
Datos de entrada:	-Tabla tabla1 -Tabla tabla2 -boolean rutas2 -String aerolineasFiltro -String modelosFiltro
Actuación:	Calcula el numero de aviones que contiene cada una de las tablas, tiene en cuenta el filtro
Funciones a las que llama:	-
Datos Salida:	-int[ ] numaviones

Función:	actualizarVentana
Quien la llama:	-Botón analizar
Datos de entrada:	-Valores de la interfaz
Actuación:	Guarda los parámetros de la ventana en la memoria
Funciones a las que llama:	-
Datos Salida:	-int starttimeVentana -int finishtimeVentana

Función:	rellenarPanelDatos
Quien la llama:	-Botón analizar
Datos de entrada:	-Valores analizados
Actuación:	Rellenamos el panel de la interfaz con los datos analizados
Funciones a las que llama:	-resetPanelDatos
Datos Salida:	-Escritura en interfaz

Función:	resetPanelDatos
Quien la llama:	-Botón Clear panel datos -función rellenarPanelDatos
Datos de entrada:	-
Actuación:	Limpiamos el panel de la interfaz
Funciones a las que llama:	-
Datos Salida:	-Escritura en interfaz

Función:	cortarRuta
Quien la llama:	-función serRuta
Datos de entrada:	-String r
Actuación:	Elimina el nombre del documento de la ruta, para poder escribir el nombre del documento que queremos abrir, sin pedir todo el historial de carpetas.
Funciones a las que llama:	-
Datos Salida:	-String rutacarpeta