Agent-Based Airport Traffic Control

Agents and Distributed Artificial Intelligence

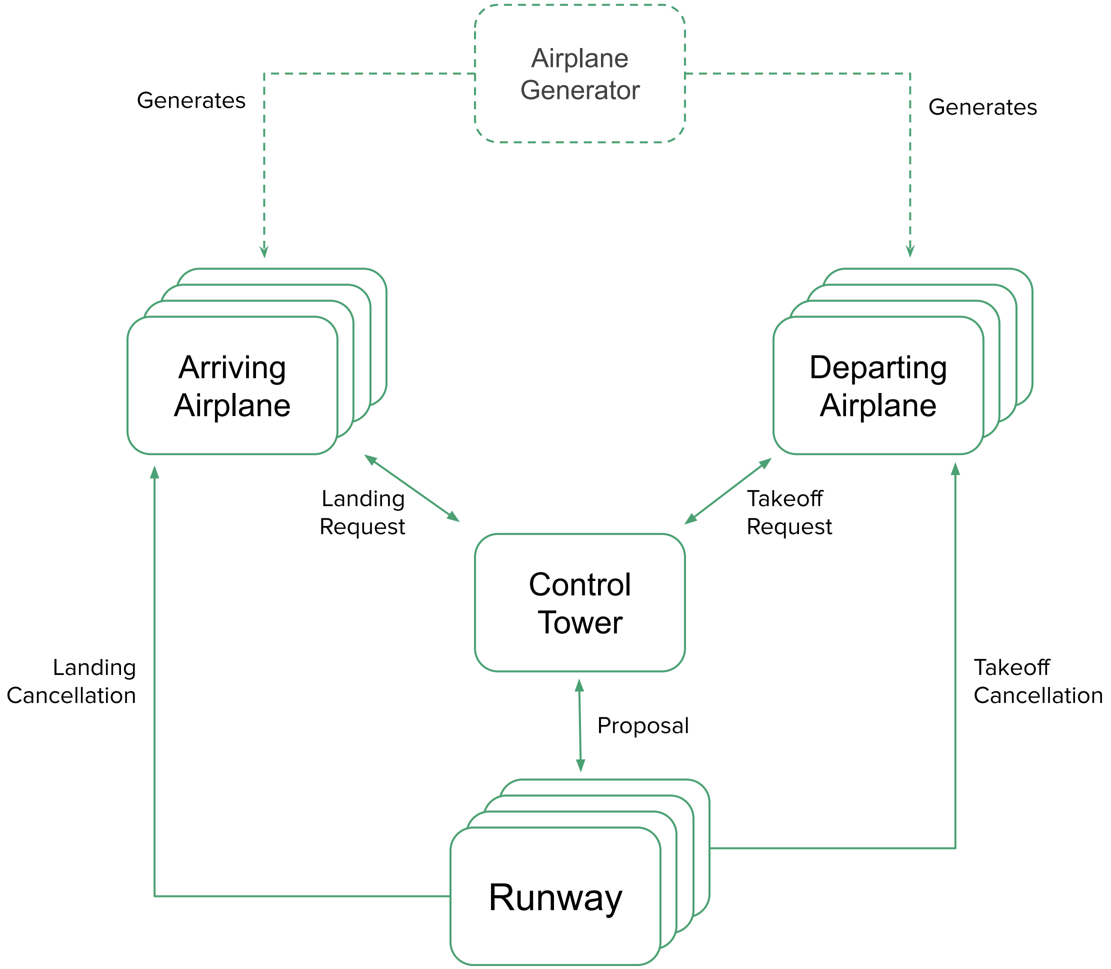
Integrated Masters, Informatics and Computing Engineering
FEUP-EIC0033-2020/2021-1S

Bernardo Costa Moreira - 201604014
Bernardo Santos - 201706534
João Nuno Matos - 201705471

Problem Description

- Aim: tackle the allocation of takeoffs and landings in a multi-runway airport
- Use the interaction between several simpler agents to achieve a complex and distributed decision-making process
- Incoming airplanes (characterized by their ETA and autonomy) want to land
- Departing airplanes (characterized by time to readiness) want to take off
- Operations must be scheduled among several runways
- A runway can become obstructed, and affected operations must be rescheduled

Agents

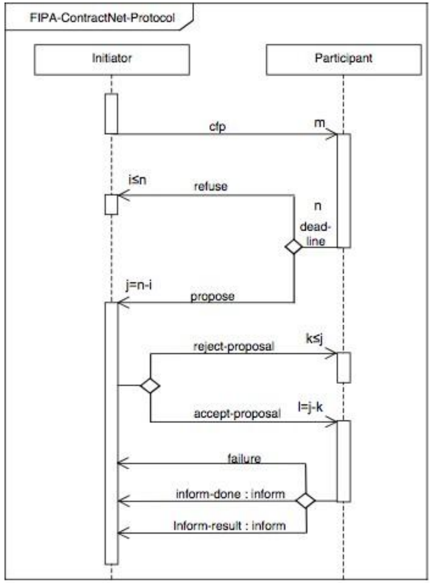

Airplane Generator
Generates
Generates
Arriving Airplane
Departing Airplane
Landing Request
Takeoff Request
Control Tower
Landing Cancellation
Takeoff Cancellation
Proposal
Runway

Protocols

Control Tower ↔ Runway(s): FIPA ContractNet

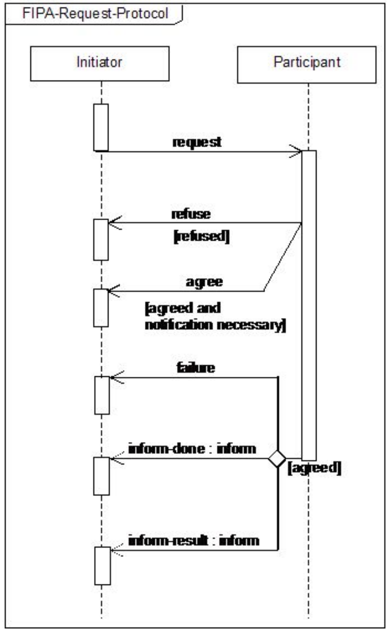
Taking ETA or time to readiness as minimum

control tower calls for proposals
each runway proposes operation schedule
control tower picks lowest waiting time



Airplane(s) ↔ Control Tower: FIPA Request

airplane requests takeoff/landing
tower procures schedule with best waiting time
if not enough autonomy, tower rejects request



Agent Architecture & Strategy

- Classes extending jade.core.Agent
- Use TickerBehaviours with uniform tick rate to control simulation speed / variables / inner state transitions
- Use Behaviours from jade.proto to implement FIPA Protocols
- JavaBeans POJOs & (get|set)ContentObject for message formatting

Agent Architecture & Strategy up close: Runways

Aims: read/update operation schedules in fast & efficient way; answer CFPs from control tower; deal with obstructions

Naïve approach: table

Time from now	0	1	2	3	4	5	6	7	8	9	10
Operation	!	!		2	2	2	3	3	3		6

Our approach: ordered map

Obstructed?	Yes; duration → 2	
Start Time	**Airplane ID**	**Duration**
3	2	3
6	3	3
10		

Dealing with obstructions ➔ cancel any affected operations and inform airplanes; they can ask for a reschedule

Yellow Pages

- Control Tower registers service with the Director Facilitator
- Control Tower subscribes for changes to existing runways
- Runways register service with DF
- Airplanes use DF to find control tower to send takeoff/landing requests

Experiments and Result Analysis

Creation Rate(s) | Obstruction Probability(%) - (Defined stop value at 30 airplanes

Important Code Variables Values : *MAX_TIME_TO_ARRIVE = 45 && MAX_FUEL_REMAINING = 15*

	10 | 0.0%	10 | 0.3	3 | 0.0	3 | 0.3
total arrivals	18	18	6	8
total_departures	12	9	13	13
total_redirects	0	6	11	12
avg_waitTimeArrival(ticks)	0	0	3	0
avg_waitTimeDepartures (ticks)	0	1	15	10

Obs: The sum of totals might not be 30 because of rescheduling;
redirected airplanes may send a new request, and they might be accepted..

Experiments and Result Analysis

In order to analyze the results, we are creating two log files, one of which to record every airplane created, and another to record operations.

We have carried out some experiments by changing the airplane creation rate and probability of obstructions.

We could conclude that an increase in traffic, proxied by a more frequent airplane creation rate, and a higher obstruction rate result in a marked increase both in the number of redirects of airplanes and in wait times.

Conclusion / Further Work

This project illustrated a use case where a multi-agent system, such as JADE, can be of great help. By using JADE, we were able to implement simple agents and behaviours that could, by following standard protocols and exchanging messages, trigger an emergent collective behaviour, while avoiding accidental complexity derived from shared state.

Compared to the original proposition, we did not implement a system to manage the airport's capacity, which is a plausible enhancement. Code organisation could stand to be improved in some parts of the code. To increase portability of agent interaction, a transition could be done from JavaBeans objects to ontologies.

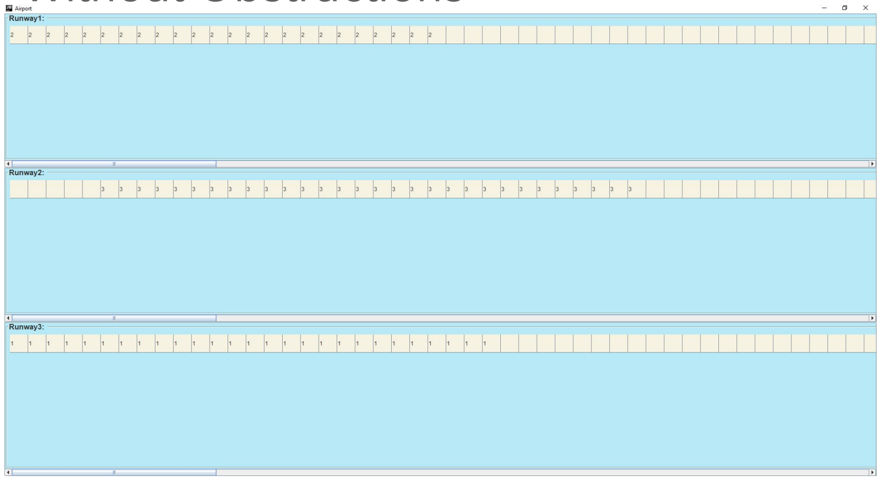
Additional Slides

GUI Examples

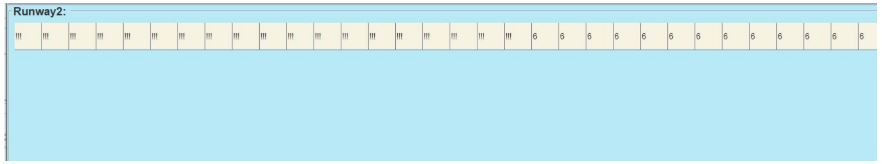
We have implemented a Graphical User Interface in order to provide a better understanding of the current state of each runway.

The GUI shows the operations that are scheduled for each runway (normal scenario, top image) including when an obstruction is present (bottom image).

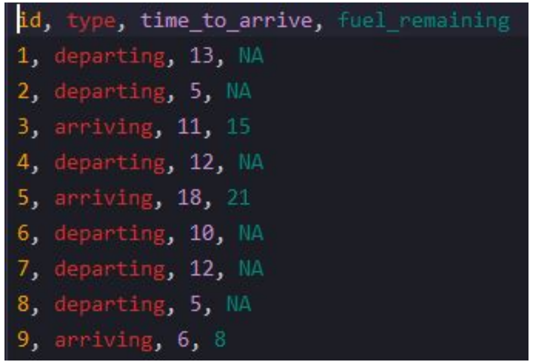
Without Obstructions



Without Obstructions



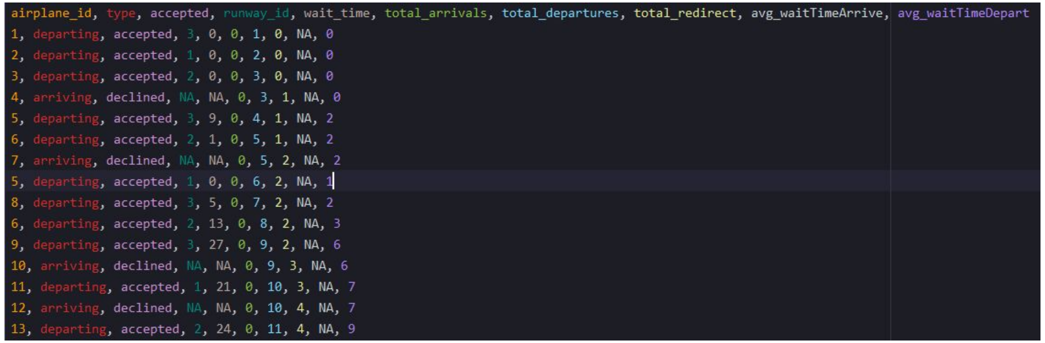
Log Examples



```
id, type, time_to_arrive, fuel_remaining
1, departing, 13, NA
2, departing, 5, NA
3, arriving, 11, 15
4, departing, 12, NA
5, arriving, 18, 21
6, departing, 10, NA
7, departing, 12, NA
8, departing, 5, NA
9, arriving, 6, 8
```

Airplane creation log: record every airplane created

- id
- type (either arriving or departing)
- estimated time of arrival (or readiness for takeoff)
- autonomy (if plane is arriving)



```
airplane_id, type, accepted, runway_id, wait_time, total_arrivals, total_departures, total_redirect, avg_waitTimeArrive, avg_waitTimeDepart
1, departing, accepted, 3, 0, 0, 1, 0, NA, 0
2, departing, accepted, 1, 0, 0, 2, 0, NA, 0
3, departing, accepted, 2, 0, 0, 3, 0, NA, 0
4, arriving, declined, NA, NA, 0, 3, 1, NA, 0
5, departing, accepted, 3, 9, 0, 4, 1, NA, 2
6, departing, accepted, 2, 1, 0, 5, 1, NA, 2
7, arriving, declined, NA, NA, 0, 5, 2, NA, 2
5, departing, accepted, 1, 0, 0, 6, 2, NA, 1
8, departing, accepted, 3, 5, 0, 7, 2, NA, 2
6, departing, accepted, 2, 13, 0, 8, 2, NA, 3
9, departing, accepted, 3, 27, 0, 9, 2, NA, 6
10, arriving, declined, NA, NA, 0, 9, 3, NA, 6
11, departing, accepted, 1, 21, 0, 10, 3, NA, 7
12, arriving, declined, NA, NA, 0, 10, 4, NA, 7
13, departing, accepted, 2, 24, 0, 11, 4, NA, 9
```

Operation log: record operation requests and running aggregate statistics

- airplane id / type of operation
- whether the request is accepted
- allocated runway
- wait time for operation
- totals for (un)successful operations
- average wait times

Detailed Execution Examples

To run our project (we recommend opening it in IntelliJ IDEA and hitting run but if not):

- Compile source files from /src (configure simulation constants in com.aiad.Config)
- Make sure JADE is in class path
- Run com.aiad.JADELaucher

To observe results

- Each table in UI is a runway's schedule (each cell is a tick), which can be empty (no operation at that time), number (operation airplane of that id), or '!' (obstructed)
- Observe debug output in stdout but prepare to be overwhelmed
- Tail log files, they are located in the working directory and are called 'allocation_log.csv' and 'creation_log.csv'

Implemented Classes

Agents:

- Airplane
- AirplaneGenerator
- ArrivingAirplane
- DepartingAirplane
- ControlTower
- Runway

Messages:

- AirplaneInform
- AirplaneRequest
- ArrivingAirplaneRequest
- RunwayOperationCfp
- RunwayOperationProposal

Others:

- Config
- JADELauncher

Other Observations

In order to make it easy to change the variables that influence the program's execution we have created the class Config, that contains the following constants:

- **TICKRATE**: how many ticks per second
- **CREATION_RATE**: seconds between each created airplane
- **MAX_TIME_TO_ARRIVE**: maximum amount of ticks an airplane may take to arrive
- **MAX_FUEL_REMAINING**: maximum amount of fuel for arriving airplanes
- **OPERATION_LENGTH**: how many ticks an airplane takes to land/takeoff
- **MAX_RUNWAY_CLEARANCE_TIME**: maximum amount of ticks for the runway to remove an obstruction
- **DEBRIS_APPEARANCE_PROBABILITY**: the probability of the runway becoming obstructed after an operation has finished