Recovering from Airline Operational Problems with a Multi-Agent System: a Case Study

António Mota[1], António J.M. Castro[1], Luís Paulo Reis[1]

[1] LIACC-NIAD&R, FEUP, Department of Informatics Engineering, University of Porto, 4200-465 Porto, Portugal
{amota, ajmc, lpreis}@fe.up.pt

Abstract. The Airline Operations Control Centre (AOCC) tries to solve unexpected problems during the airline operation. Problems with aircraft, crewmembers and passengers are common and very hard to solve due to the several variables involved. This paper presents the implementation of a real-world multi-agent system for operations recovery in an airline company. The analysis and design of the system was done following a GAIA based methodology. We present the system specification as well as the implementation using JADE. A case study is included, where we present how the system solved a real problem.

Keywords: Disruption management, operations recovery, multi-agent system.

1 Introduction

The disruption management of an airline operation is a process by which problems that might affect the proper implementation of the operational scheduling of the company (problems related with crewmembers, aircraft, and/or passengers of a flight), are solved. This process is based on three key components: monitoring (supervising the airline operation in a particular base), event detection (identification of situations that could put at risk the operational planning), and resolution of problems (identify solutions that can mitigate the problems encountered). Since this is a very important process for the airline, it usually has a department responsible for its implementation: the Operational Control Centre (OCC). In this centre, operating 24 hours a day, working groups of people are responsible for monitoring the operation, detecting events, and proposing solutions to the Supervisor of the OCC, an entity that decides whether the solution will be implemented or not. Those solutions are achieved based mostly on the tacit knowledge of the people and there is no automated manner of how to address the resolution of the problems. Although there are software tools that help the elements of the OCC throughout the process (especially at the stage of monitoring and when obtaining information useful to solve the various problems), they are often obsolete, do not cover the whole process, and are not integrated. The work we presented here consists on the development of a Multi-Agent System (MAS)

representing the OCC and the various elements existing there. The resolution of the problems will be made by a group of agents, a process known as disruption management. These agents use meta-heuristics, although the system is designed to permit the inclusion of an unlimited number of agents that can use any method of problem resolution. This system was developed in an airline company and the motivation to develop it came from the observation of the needs of that company OCC. It was thus clear that there was a good opportunity to develop a system that, in some way, could catch and use the knowledge of the elements of the OCC, to make the process automatic, or at least to automate some of its more repetitive tasks.

Although there is extensive research applied to problems of scheduling on airlines (see, for example, [1], [3], [5] and [11]), there are very few works of really functional MAS on real airlines, capable of an integrated resolution of the type of problems that are found on the OCC. Therefore, our system would be innovative, useful not only for the company in question, but potentially also for others. Additionally, it is important to highlight the economic value that comes from the use of a system like this. First, by automating part of, or even all, the process, it is no longer required the presence of so many elements in the OCC, which lets the company reduce costs with staff; this reduction is even more evident because the OCC is a department that operates 24 hours a day. Furthermore, the use of Artificial Intelligence methods to resolve the various problems (flight delays, missing crewmembers, etc.) can contribute to the improvement of the quality of the solutions implemented, which translates into a decrease in costs associated with the modification of the operation of the airline company.

The MAS should monitor the operation that is being conducted on each operational base of the company (that is, the flights, the crewmembers who have reported for duty, etc.) detecting events that may correspond to problems that need to be resolved (some minor events can be skipped). Some of the possible events are: aircraft failures, flight delayed due to bad weather and / or congestion of air traffic, shortages of crewmembers, etc. The system should permit the definition of what is an event (a ten minutes delay of a crewmember can be a problem for one airline, but not for another). The system must have a set of agents, cooperating with each other, that are specialists in the resolution of the various types of problems. According to [6] the presence of agents that implement different methods for solving the same problem will increase the robustness of the system because of this redundancy. Finally, the system should have a visual interface that allows the supervisor to monitor the operation, to detect the events and to observe the solution proposed for the specific problem at hand.

It is important to point out that the analysis and design of this system followed a GAIA [14], [4] based methodology. For the interested reader, please see [7] for a comparison of several agent-oriented methodologies available. The rest of the paper is organized as follows. Section 2 presents some related work. Section 3 details the system specification and implementation, including the architecture, use cases and details about the specialist agents' implementation. Section 4 shows a case study and, in section 5, we conclude by discussing our work and presenting future implementations.

2 Related work

Aircraft Recovery: Liu et al. [9] proposes a "multi-objective genetic algorithm to generate an efficient time-effective multi-fleet aircraft routing algorithm" in response to disruption of flights. It uses a combination of a traditional genetic algorithm with a multi-objective optimization method, attempting to optimize objective functions involving flight connections, flight swaps, total flight delay time and ground turn-around times. According to the authors "(…) the proposed method has demonstrated the ability to solve the dynamic and complex problem of airline disruption management".

Crew Recovery: In Abdelgahny et al. [1] the flight crew recovery problem for an airline with a hub-and-spoke network structure is addressed. The paper details and sub-divides the recovery problem into four categories: misplacement problems, rest problems, duty problems, and unassigned problems. Due to the stepwise approach, the proposed solution is sub-optimal. According to the authors the tool is able to "solve for the most efficient crew recovery plan with least deviation from originally planned schedule".

Integrated Recovery: Bratu et al. [3] presents two models that considers aircraft and crew recovery and through the objective function focuses on passenger recovery. They include delay costs that capture relevant hotel costs and ticket costs if passengers are recovered by other airlines. The objective is to minimize jointly airline operating costs and estimated passenger delay and disruption costs. According to the authors, "(…) decisions from our models can potentially reduce passenger arrival delays (…) without increasing operating costs".

Other Application Domains: Agents and multi-agent systems have been applied both to other problems in air transportation domain and in other application domains. A brief and incomplete list of such applications follows. Tumer and Agogino [12] developed a multi-agent algorithm for traffic flow management. For ATC Tower operations, Jonker et al. [8] have also proposed the use of multi-agent systems. As a last example, a multi-agent system for the integrated dynamic scheduling of steel production has been proposed by Ouelhadj [10].

3 System Specification and Implementation

System overview: Figure 1 shows the overall architecture of the multi-agent system.

There are four types of agents:

- *Monitor*, which monitors the operation of the airline company.
- *Event Detector*, which defines the type of events that must be detected.
- *Resolution Manager*, which receives a problem and manages the resolution in cooperation with the specialist agents.
- *Specialist*, which is responsible for the resolution of a problem using a specific method.

The communication between the agents is done through the JADE system [2]. Additionally, Figure 1 shows the existence of a data store, that has information about

the airline company operations. That data store is accessed by the *monitor* and by the *specialist* agents. Figure 1 also shows the presence of a human: the OCC supervisor.

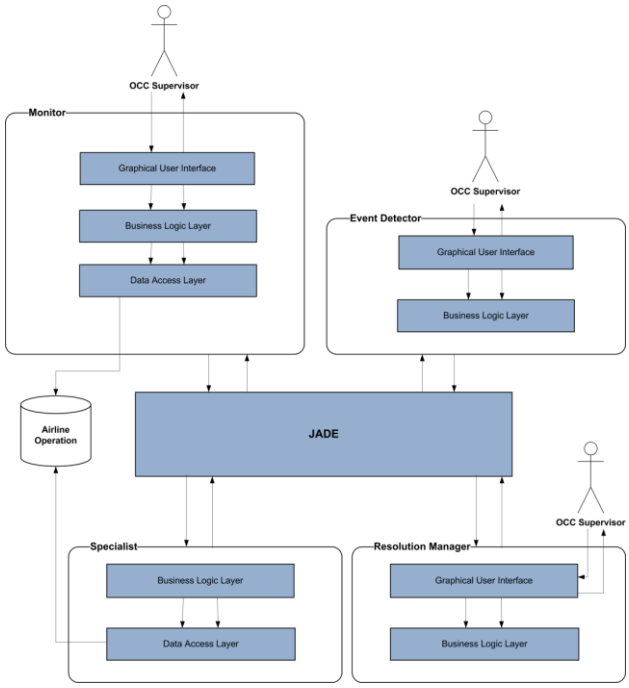

Fig. 1. Overall architecture of the Multi-Agent System

The OCC supervisor starts the monitoring process and defines the type of events that are to be considered warnings and problems. Then, the event detector agent communicates those types of events to the monitor, which shows the warnings and/or the problems to the OCC supervisor. This entity establishes the parameters of the resolution process and decides to solve a specific problem. The problem is communicated by the monitor to the resolution manager, which in turn communicates it to all the specialist agents. After arriving at a solution, each agent sends it to the resolution manager, which picks the best, i.e., the one with less cost and shows it to the OCC supervisor.

Figure 2 shows all the use cases of the system. The JADE actor is represented because the reception of messages from other agents is periodically checked by using a type of JADE behaviour (so, the JADE actor periodically asks the system to check for new messages, acting as an actor).

In the *monitoring* subsystem, the OCC supervisor can check the details of a particular flight that is being monitored, as well as order the resolution of a specific problem. The JADE system periodically asks the system to monitor the operation and to check the arrival of *MetaEvent* messages. In the *event detection* subsystem, the OCC supervisor can manage *MetaEvents*, that is, add, remove, or modify a type of event. Each time the set of *MetaEvent* is changed, a message is sent to the monitor. Finally, in the *resolution* subsystem, the OCC supervisor can manage the solving

parameters and check the solution. The JADE system periodically asks the system to check the arrival of problem messages (which triggers the resolution of that problem), and solution messages (which triggers its presentation to the OCC supervisor).

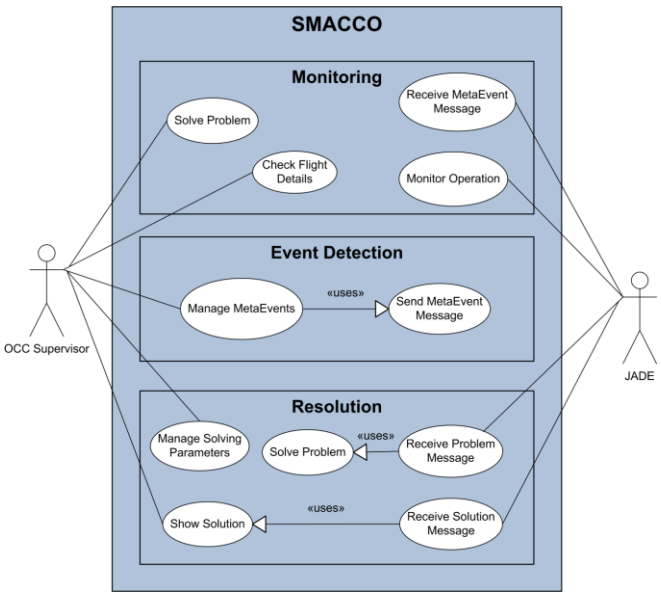

Fig. 2. System use cases

Monitoring subsystem specification: In the MAS there are three different subsystems as stated previously. Due to the lack of space we are going to present the specification for two of them: *Monitoring* and *Resolution* subsystem. Table 1 describes the use case "*Monitor Operation*".

Table 1. Use case "Monitor Operation"

UC ID:	SMACCO-BUC001.00	
Name:	**Monitor Operation**	
Date:	10/12/2007	
Actors:	JADE	
Trigger:	This use case is initiated when the JADE cyclic behaviour triggers its action (the operation monitoring).	
Desc:	1. Obtain flights that are going to arrive within a specific amount of time. 2. Obtain flights that arrived within a specific amount of time ago. 3. Obtain flights that are going to depart within a specific amount of time. 4. Obtain flights that departed within a specific amount of time ago. 5. Signal the flights on the interface, separating the NB and WB flights. 6. Check if there are types of events to monitor 7. If there are types of events to monitor, check what flights (if any) match the conditions of the type of event. 8. If there are flights that correspond to the type of event conditions, signal the type of event in the interface.	

The monitor agent implements five behaviours:

- A *TickerBehaviour* that periodically accesses the company's operational database to monitor the operation.
- A *CyclicBehaviour* that waits for a meta event message to arrive, using a *ReceiverBehaviour* to filter the appropriate messages.
- An *OneShotBehaviour* that sends a message with a set of events to be solved.
- A *ParallelBehaviour* that execute the former behaviours concurrently.

Monitoring subsystem implementation: The JADE implementation of the *BehaviourMonitor* class, which extends *TicketBehaviour*, is presented below. The method *onTick* contains the piece of code that will be executed periodically.

```
Class BehaviourMonitor extends TickerBehaviour {
  public BehaviourMonitor(Agent agent, long period)
  { super(agent, period); }
  @Override
  protected void onTick() {
     try {
        Monitor('NB');
        Monitor('WB');
        if(eventsDefinition.size() > 0)
          markEvents();
     }
     Catch (SQLException ex) {
        ex.printStackTrace(); }
  }}
```

The agent periodically monitors the narrow body (an aircraft with a single aisle) and wide body (an aircraft with two aisles) airline operation and, if there are any type of events to be detected, checks if any flight is under the conditions necessary to consider itself a warning or a problem. The *BehaviourProblemResolution* class, which extends *OneShotBehaviour*, is presented below. The piece of code inside the *action()* method will be executed only once.

```
class BehaviourProblemResolution extends OneShotBehaviour {
  public BehaviourProblemResolution(Agent agent)
  { super(agent); }
  @Override
  protected void action() {
     ACLMessage msgToSend = prepEventsMsg(events,
Shared.getAgents(super.myAgent, 'AgentManager'));
     send(msgToSend);
  }}
```

In this behaviour, a message is sent from the monitor agent to the resolution manager containing the list of events that triggered the problem.

Resolution subsystem specification: This subsystem receives a problem and manages the resolution in cooperation with the specialist agents. The resolution manager agent implements five behaviours:

- A *CyclicBehaviour* that waits for a problem message to arrive, using a *ReceiverBehaviour* to filter the appropriate messages.
- A *CyclicBehaviour* that waits for a solution message to arrive, using a *ReceiverBehaviour* to filter the appropriate messages.

– A *ParallelBehaviour* that executes the former behaviours concurrently.
– A *SimpleBehaviour* that sends a message with the problem to be solved to the specialist agents.

Each specialist agent implements
– A *CyclicBehaviour* that waits for a problem message to arrive, using a *ReceiverBehaviour* to filter the appropriate messages.
– A *ParallelBehaviour* that executes the former behaviours concurrently.

Two specialist's agents were implemented, each one solving the problem via different meta-heuristics: *hill-climbing* and *simulated annealing*. The *hill climbing agent* solves the problem iteratively by following the steps:

1. Obtains the flights that are in the time window of the problem. This time window starts at the flight date and ends at a customizable period in the future. This will be the initial solution of the problem. The crewmembers and aircraft exchanges are made between flights that are inside the time window of the problem.
2. While some specific and customizable time has not yet passed or a solution below a specific and customizable cost has not been found, repeats steps 3 and 4.
3. Generates the successor of the initial or previous solution.
4. Evaluates the cost of the solution according to Equation 3. If it is smaller than the cost of the current solution, accepts the generated solution as the new current solution. Otherwise, discards the generated solution. The way a solution is evaluated is described below.
5. Send the current solution to the agent that manages the resolution process.

The *simulated annealing agent* solves the problem iteratively by following the steps:

1. Obtains the flights that are in the time window of the problem.
2. While some specific and customizable time has not yet passed or a solution below a specific and customizable cost has not been found, repeats steps 3 and 4.
3. Generates the successor of the initial or previous solution.
4. Evaluates the cost of the solution according to Equation 3. If it is smaller than the cost of the current solution, accepts the generated solution as the new current solution. Otherwise, it accepts the generated solution with a probability that is given by Equation 1.

$$P = \exp\left(-\Delta E/T\right) \tag{1}$$

Where ΔE is the difference between the cost of the generated solution and the cost of the current one, and T is given by

$$T_{t+1} = \alpha T_t \tag{2}$$

Where α lies between 0 and 1. α was given a value of 0.8, and T was given an initial value of 10. T is updated every N iterations. N was given a value of 2.

5. Send the current solution to the agent that manages the resolution process.

Resolution subsystem implementation: The implementation of the hill-climbing algorithm on one of the specialist agents, using JADE, follows:

```
while (!Shared.para(problem.getNumSeconds(), secondsExecution,
problem.getMaximumCost(), costActualSolution)) {
  // get sucessor
  successor.Shared.newSucessor(Shared.copyArrayList(currentSolution);
  costSucessor=Shared.calcCost(sucessor, plainInitialSolution);
  if (costSucessor < costActualSolution) {
  currentSolution = sucessor;
  costActualSolution = costSucessor;
  }}
```

Solution generation and evaluation: The generation of a new solution is made by finding a successor that distances itself to the current solution by one unit, that is, the successor is obtained by one, and only one, of the following operations:

- Swap two aircrafts between flights that belong to the flights that are in the time window of the problem.
- Swap two crewmembers between flights that belong to the flights that are in the time window of the problem.
- Swap an aircraft that belongs to the flights that are in the time window of the problem with an aircraft that is not being used.
- Swap a crewmember of a flight that belongs to the flights that are in the time window of the problem with a crewmember that isn't on duty, but is on standby.

When choosing the first element (crewmember or aircraft) to swap, there are two possibilities:

- Choose randomly.
- Choose an element that is delayed.

This choice is made based on the probability of choosing an element that is late, which was given a value of 0.9, so that the algorithms can proceed faster to good solutions (exchanges are highly penalized, so choosing an element that is not late probably won't reduce the cost, as a possible saving by choosing a less costly element probably won't compensate the penalization associated with the exchange). If the decision is to exchange an element that is delayed, the list of flights will be examined and the first delayed element is chosen. If the decision is to choose randomly, then a random flight is picked, and a crewmember or the aircraft is chosen, depending on the probability of choosing a crewmember, which was given a value of 0.85. When choosing the second element that is going to swap with the first, there are two possibilities:

- Swap between elements of flights
- Swap between an element of a flight and an element that isn't on duty.

This choice is made based on the probability of choosing a swap between elements of flights, which was given a value of 0.5. The evaluation of the solution is done by an objective function that measures four types of costs:

- The costs with crewmembers. Those costs take into consideration the amount that has to be paid to the crewmember (depends on the duration of the flight), and the

base of the crewmember (for instance, assigning a crewmember from Oporto to a flight departing from Lisbon has an associated cost that would not be present if the crewmember's base was Lisbon).

- The costs with aircrafts. Those costs take into consideration the amount that has to be spent on the aircraft (depends on the duration of the flight), and the base where the flight actually is.
- The penalization for exchanging elements.
- The penalization for delayed elements. The cost associated with this aspect is the highest, because the goal is to have no delayed elements.

These types of costs are taken into account in Equation 3:

$$tc = cmc + amc + exW * numE + dlW + numD \quad (3)$$

Where

$$cmc = \sum_{i=1}^{|CM|} (c_i * bcf)/numCm \quad (4)$$

where
$i \in CM; CM = \{all\ crewmembers\ in\ flight\}$
$1 < bcf \leq 2 : base\ crew\ factor$
$numCm : number\ of\ crew\ members\ in\ CM$

$$amc = \sum_{j=1}^{|AC|} (ac_j * baf)/numAc \quad (5)$$

where
$j \in AC; AC = \{aircraft\ same\ fleet\}$
$1 < baf \leq 2 : base\ aircraft\ factor$
$numAc : number\ of\ aircrafts\ in\ AC$

ExW was given a value of 1000, and *DlW* a value of 20000.

4 Case study

This scenario is based on the delay of flight TP935 at 08/04/2008 17:20, from Lisbon (LIS) to Paris (CDG) with a schedule departure time of 17:05 and expected arrival time of 18:41. There were 15 business class passengers and 113 economy class passengers. Figure 3 shows the graphical user interface of the monitoring and event detection system. At the upper half of the interface: the flights that are arriving (in this case, is the next 80 minutes), that already arrived (in this case, in the next 15 minutes), that are departing (in this case, in the next 45 minutes), and that already have departed (in this case, until 40 minutes ago) are shown. The flight TP935 is, as expected, in the group of the flights that will depart. At the bottom half of the interface, details are shown about the flight, its crewmembers, and the aircraft used. The monitoring system is constantly monitoring the operation, so that the information presented in the interface is maintained actual. As can be seen at the figure, the system shows seven warnings and five problems. The flight TP2676 shows a warning

because the flight should have arrived more than 15 minutes ago and, as a consequence, all the passengers are more than 15 minutes late. The same happens with flight TP432, that should have departed more than 10 minutes ago, which again makes all the passengers in it behind schedule.

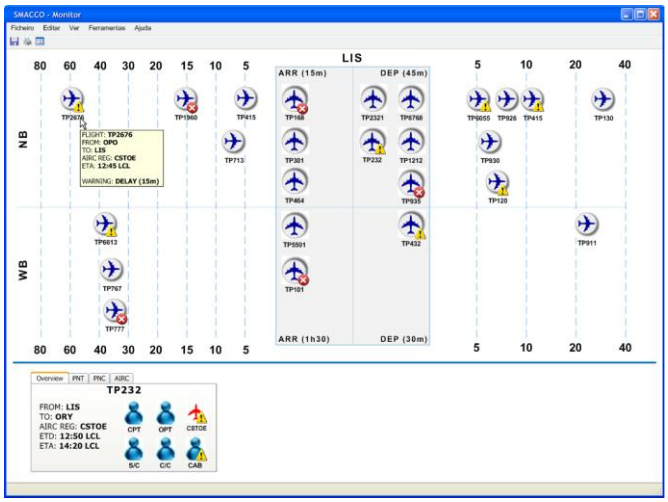

Fig. 3. Monitoring and Event Detection System

Finally, according to the system, the operation of the airline company has one warning and four problems on flight TP935:

- The flight is 20 minutes late and, therefore, that triggers a warning.
- Two crewmembers are more than 15 minutes late and that triggers two problems. Those crewmembers are late because they are on flight TP2676, which is late.
- Another crewmember is more than 10 minutes late and that triggers a problem. That crewmember did not report for duty.
- Aircraft CSTOE is more than 15 minutes late and that triggers a problem. The aircraft is late because it is the aircraft of flight TP2676, which is late.

The OCC personnel are now able to solve the problem of flight TP935 in cooperation with the problem resolution system.

Problem resolution: The system shows the description of the problem and some of the parameters that can be adjusted by the OCC personnel. For example, available time to find a solution and maximum allowed cost of the solution. Those parameters will be used when finding a solution.

The resolution process is transparent to the OCC personnel, as they don't know how many agents participated on the process and what methods did they use. The management of the process is the responsibility of an agent, which chooses the best solution among the ones given (the one with less cost according to Equation 3), and presents that solution to the OCC personnel. In this case, the solution was:

– Switch aircrafts CSTOE and CSTNJ between flight number TP935 and flight number TP1212. Because TP1212 will depart later than TP935, the aircraft CSTOE has good chances of arriving before the flight's schedule departure date.
– Switch three crewmembers between flight number TP935 and flight number TP1212.

The solution to the problem has a lower cost than to do nothing, especially because of the high penalizations associated with crew and aircraft delays. In this case, the initial solution has a cost of 98340 and the proposed by the system has a cost of 26218.

5 Conclusions and Future Work

We conclude that the goals established for this project have been achieved and that the basis for its possible near future use in the airline company was gotten underway. Indeed, it is an integrated system that automates much of the disruption management process, from the monitoring of the operation of the company in its several bases, to the detection of events and the resolution of the problems encountered. Additionally, our MAS is oriented to the future: its distributed nature and the fact that it is based on agents that are specialists in solving problems easily allows the insertion into the system of new agents that solve new kinds of problems that were identified in the meantime, or that resolve the current types of problems using different methods. It is thus a truly scalable solution, prepared to sustain the growth of the airline company.

Of course, some difficulties arose during the development of the project. Obtaining a graphical interface that would allow the OCC personnel to view, in an easy and efficient way, the operation of the company at a particular base was a task that involved a lot of effort, with the need to refine it over time. Cooperation with some elements of the airline company was crucial in this process. With regard to the obtaining of the information, the analysis of the database that contains information on the operation of the airline was an arduous task, given its complexity and the strong presence of specific language to the domain of aviation. Finally, some difficulties have arisen in the fine-tuning of the objective function of the various meta-heuristics used, particularly in the choice of the weights to be given to each factor. Nonetheless, the number of experiments performed enabled reaching values that generated quite acceptable solutions.

From the case study presented we cannot assume that our system will always have this behaviour in all situations. For that, we need to perform several tests using simulated and/or real events or, if possible, running our system in parallel with the current system at the airline company. Due to the probabilistic nature of the simulated annealing algorithm and without performing more tests, we cannot generalize the results presented here.

Although the goals have been achieved, it is important to consider a number of improvements that could be made on future developments and that could enrich it. In terms of the algorithms used to solve the problems, other meta-heuristics can be implemented, as well as methods based in the area of operational research

Another useful improvement for the system would be to provide it with the ability to implement the solution, which currently is made by the supervisor of the OCC. Despite being somewhat technologically accessible, this add-on to the system should be the subject of special care because of the critical nature of the operation.

Acknowledgments. The second author is supported by FCT (Fundação para a Ciência e Tecnologia) under research grant SFRH/BD/44109/2008. The authors are grateful to TAP Portugal for allowing the use of real data from the company.

References

1. Abdelgahny, A., Ekollu, G., Narisimhan, R., and Abdelgahny, K.: A Proactive Crew Recovery Decision Support Tool for Commercial Airlines during Irregular Operations. Annals of Operations Research, 127:309-331 (2004).
2. Bellifemine, F., Caire, G., Trucco, T., and Rimassa, G.: JADE Programmer's Guide. JADE 3.3 TILab S.p.A. (2004).
3. Bratu, S., and Barhhart, C.: Flight Operations Recovery: New Approaches Considering Passenger Recovery. Journal of Scheduling 9(3):279-298 (2006).
4. Castro, A. and Oliveira, E.: The rationale behind the development of an airline operations control centre using Gaia-based methodology. Int. J. Agent-Oriented Software Engineering, Vol. 2, No. 3, pp.350–377 (2008).
5. Clausen, J., Larsen, A., and Larsen, J.: Disruption Management in the Airline Industry – Concepts, Models and Methods. Technical Report, 2005-01, Informatics and Mathematical Modeling, Technical University of Denmark, DTU (2005).
6. Castro, A.J.M., and Oliveira, E.: Using Specialized Agents in a Distributed MAS to Solve Airline Operations Problems: a Case Study. Proceedings of IAT 2007 (Intelligent Agent Technology Conference), pp. 473-476, Silicon Valley, California, USA 2-5 November 2007, IEEE Computer Society, ISBN: 0-7695-3027-3.
7. Henderson-Sellers, B, and Giorgini, P. (eds): Agent-Oriented Methodologies. Idea Group Publishing, ISBN: 159140586-6 (2005)
8. Jonker, G., Meyer, J.J., and Dignum, F.: Towards a Market Mechanism for Airport Traffic Control, 12th Portuguese Conference on Artificial Intelligence (EPIA 2005), Covilhã, Portugal (2005).
9. Liu, T., Jeng, C., and Chang, Y.: Disruption Management of an Inequality-Based Multi-Fleet Airline Schedule by a Multi-Objective Genetic Algorithm, Transportation Planning and Technology, Vol. 31, No. 6, pp. 613-639 (2008).
10. Ouelhadj, D.: A Multi-Agent System for the Integrated Dynamic Scheduling of Steel Production, Ph.D. dissertation, The University of Nottingham, School of Computer Science and Information Technology, England (2003).
11. Rosenberger, J., Johnson, E., and Nemhauser, G.: Rerouting aircraft for airline recovery. Technical Report, TLI-LEC 01-04, Georgia Institute of Technology (2001).
12. Tumer, K., and Agogino, A.: Distributed Agent-Based Air Traffic Flow Management, 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007), Honolulu, Hawaii (2007).
14. Zambonelli, F., Jennings, N. and Wooldridge, M.: Developing multi-agent systems: the Gaia methodology', ACM Transactions on Software Engineering and Methodology, Vol. 12, No. 3, pp.317–370 (2003).