

UNIVERSITY OF PORTO
FACULTY OF ENGINEERING



**Designing a Multi-Agent System for Monitoring and Operations Recovery
for an Airline Operations Control Centre**

by

António Jesus Monteiro de Castro

Graduate in Information Systems Engineering
By ISEP - Institute of Engineering of Porto

Submitted to the Department of Electrical Engineering and Computers
In partial fulfillment of the requirements for the Master Degree
in Artificial Intelligence and Intelligent Systems

Thesis supervised by

Professor Eugénio Oliveira (Phd)
Faculty of Engineering, University of Porto and LIACC

December 2006

Versão 1.1 (22ABR07)

ERRATA

RESUMO

A operação de uma companhia aérea raramente acontece como planeado. São comuns os problemas relacionados com os aviões, com as tripulações e com os passageiros. As acções que têm como objectivo resolver estes problemas são conhecidas como Gestão das Irregularidades Operacionais. O Centro de Controlo Operacional da Companhia Aérea (CCO) tenta resolver estes problemas com o mínimo de impacto na operação, com o mínimo custo e, ao mesmo tempo, satisfazendo todas as regras de segurança requeridas. Normalmente, cada problema é tratado separadamente e algumas ferramentas têm sido propostas para ajudar no processo de tomada de decisão pelos coordenadores destes centros de controlo.

Observamos o CCO da TAP Portugal, a maior companhia aérea Portuguesa, e, destas observações, várias hipóteses foram identificadas e algumas experimentadas. Acreditamos, e esta é uma das nossas principais hipóteses, que o paradigma do Sistema Multi-Agente (SMA) é mais adequado para representar a organização hierárquica de vários níveis e as várias funções (*roles*) existentes no CCO. Nesta tese, propomos o desenho e a implementação parcial de um SMA Distribuído que represente as várias funções existentes no CCO. Admitimos a hipótese de que, tirando partido do facto de que cada base operacional tem recursos específicos (quer aviões quer tripulantes) e juntando informações que digam respeito aos custos envolvidos (por exemplo, informação sobre vencimentos dos tripulantes, custos dos hotéis, entre outros), as soluções para os problemas detectados serão encontradas mais rapidamente e serão menos caras. Também admitimos a hipótese de que se utilizarmos agentes de software especializados que implementam diferentes soluções (heurísticas e outras soluções baseadas em modelos de investigação operacional e algoritmos de inteligência artificial) aplicadas ao mesmo problema, a robustez do sistema irá aumentar. Finalmente, acreditamos que a inclusão de um mecanismo de aprendizagem, que aprenda com a utilização anterior dos tripulantes, irá aumentar a qualidade das soluções. Estendendo esse mecanismo de forma a aprender o perfil de cada tripulante e aplicando esse conhecimento na geração de planeamentos (escalas) futuros, a gestão deste recurso tão caro será muito mais eficiente e o nível de satisfação de cada tripulante irá aumentar.

Apresentamos um caso de estudo real, obtido no CCO da TAP, onde um problema relacionado com tripulantes é resolvido usando o SMA proposto. Apresentamos resultados computacionais, usando uma operação real da companhia aérea, incluindo a comparação com uma solução para o mesmo problema encontrada pelo operador humano do CCO. Mostramos que, mesmo para problemas simples e quando comparado com soluções encontradas por operadores humanos, no caso específico desta companhia aérea, é possível encontrar soluções válidas, em menos tempo e com menos custos.

Nesta tese também mostramos como completamos a metodologia GAIA de forma a melhor analisar e desenhar o SMA proposto para o CCO. Para além de mostrarmos o *rationale* que está por trás da análise, desenho e implementação do nosso sistema, também mostramos como mapeamos as abstrações usadas no desenho orientado a agentes para código específico em JADE. As vantagens da utilização de uma análise de requisitos orientada a objectivos e a sua influência nas fases seguintes da análise e do desenho, também são apresentadas. Finalmente, propomos diagramas UML 2.0 para representação de vários *deliverables* da GAIA, tais como, estrutura

organizacional, modelos de funções (*role*) e de interações e modelos de agentes e de serviços.

ABSTRACT

An airline schedule seldom operates as planned. Problems related with aircrafts, crew members and passengers are common and the actions towards the solution of these problems are usually known as operations recovery or disruption management. The Airline Operations Control Center (AOCC) tries to solve these problems with the minimum impact in the airline schedule, with the minimum cost and, at the same time, satisfying all the required safety rules. Usually, each problem is treated separately and some tools have been proposed to help in the decision making process of the airline coordinators.

We have observed the AOCC of TAP Portugal, the major Portuguese airline, and, from those observations, several hypotheses have been identified and some of them experimented. We believe, and that is one of our main hypothesis, that the Multi-Agent System (MAS) paradigm is more adequate to represent the multi-level hierarchy organization and the several roles that are played in an AOCC. In this thesis we propose the design and partial implementation of a Distributed MAS representing the existing roles in an AOCC. We hypothesize that if we take advantage of the fact that each operational base has specific resources (both crew and aircrafts) and that if we include information regarding costs involved (for example, crew payroll information and hotels costs, among others), the solutions to the detected problems will be faster to find and less expensive. We also hypothesize that if we use specialized software agents that implement different solutions (heuristic and other solutions based in operations research models and artificial intelligence algorithms), to the same problem, the robustness of the system will increase. Finally, we believe that the inclusion of some kind of learning mechanism that learns from previous utilization of crew members will improve the solutions quality. Extending that learning mechanism to learn each crew member profile, and applying that knowledge for generating future schedules, the management of that expensive resource will be much more efficient and the level of satisfaction of each crew member will increase.

We also present a real case study taken from TAP Portugal AOCC, where a crew recovery problem is solved using the MAS. Computational results using a real airline schedule are presented, including a comparison with a solution for the same problem found by the human operators in the Airline Operations Control Center. We show that, even for simple problems, and when comparing with solutions found by human operators in the case of this airline company, it is possible to find valid solutions, in less time and with a smaller cost.

In this thesis we also show how we complement the GAIA methodology in order to better analyze and design the proposed MAS for the AOCC. Besides showing the rationale behind the analysis, design and implementation of our system, we also present how we mapped the abstractions used in agent-oriented design to specific constructs in JADE. The advantages of using a goal-oriented early requirements analysis and its influence on subsequent phases of analysis and design are also presented. Finally, we also propose UML 2.0 diagrams at several different levels for representation of GAIA deliverables, like organizational structure, role and interaction model, agent and service model.

PROLOGUE

It was not difficult to find the necessary motivation to do this work. During 10 years I have been the crew manager at TAP Portugal Porto base, with the responsibility for scheduling and managing the crew members of this base. In the final year of my graduation in information systems engineering, in 1994, I had the opportunity, through an ERASMUS scholarship, to do the final project in the Leeds Metropolitan University in England. The final project was the development of a “Crew Management and Planning System” using DBFAST© (a kind of DBIII language and database) in Windows 3.1©. That software was used in Porto base from 1995 to the year 2000 and also in other airline companies.

Ten years later, in 2004, when I decide to do a Master Degree in Artificial Intelligent and Intelligent Systems, I found the perfect opportunity to apply all the knowledge I have about this domain. Nowadays the technology is quite different than in 1994, given much more possibilities than ever to approach this subject. Artificial intelligence by itself is a very interesting topic with great potential to be applied in solving real world problems. The merge of my willing to learn with the knowledge I have about the domain, gave me the necessary motivation to do this work.

If it was not difficult to find the motivation, the time to do the job was much more difficult to find. A lot of weekends and national holidays were spent in writing this thesis. That time should be reserved for the family and, because of that I did not have the pleasure of benefitting from the company of my wife, Isabel, and my two daughters, Isabel and Inês. Without their patient I could not finish this thesis. So, my first word of gratitude has to go to my family: I love you and I miss the moments we did not share together.

I’m also grateful to all the colleagues at TAP Portugal, for all the help they gave to me.

A special word of gratitude goes to Eng. Manoel Torres, TAP EVP, for believing in me.

Finally, I want to thank to my supervisor, Professor Eugénio Oliveira, for all the help and counseling during this work. I hope to continue this partnership through my research work.

*Dedicated to the memory of my grandfather
António Magalhães Monteiro
Your counsels during my childhood
Are still in my memory
I never cease to follow them*

TABLE OF CONTENTS

1. INTRODUCTION	23
1.1. OVERVIEW	23
1.2. THE AIRLINE SCHEDULING PROBLEM	25
1.3. THE OPERATIONS RECOVERY PROBLEM	27
1.4. OBSERVATIONS	29
1.5. HYPOTHESIS AND PREDICTIONS	31
1.6. EXPECTED RESULTS	33
1.7. STATE OF THE ART: OPERATIONS RECOVERY	35
1.8. STATE OF THE ART: AGENT-ORIENTED METHODOLOGIES	37
2. METHODS USED	41
2.1. VISION AND SCOPE	41
2.2. METHODOLOGY, TOOLS AND TECHNIQUES	45
2.1. REQUIREMENTS AND SPECIFICATIONS	47
2.1.1. <i>Early Requirements Analysis</i>	47
2.1.2. <i>Analysis</i>	51
2.1.2.1. Organizations	51
2.1.2.2. Environment Model	52
2.1.2.3. Preliminary Role Model	56
2.1.2.4. Preliminary Interaction Model	60
2.1.2.5. Organizational Rules	62
2.1.3. <i>Architectural Design</i>	65
2.1.3.1. Organizational Structure	66
2.1.3.2. Role Model and Interaction Model	72
2.1.4. <i>Detailed Design</i>	77
2.1.4.1. Agent Model	77
2.1.4.2. Service Model	78
2.1.4.3. UML 2.0 Representation	79
2.2. IMPLEMENTATION	81
2.2.1. <i>Concepts and Actions</i>	81
2.2.2. <i>Agents, Protocols and Services Notations</i>	82
2.2.3. <i>Services and JADE behaviors</i>	84
2.2.4. <i>ACL Performatives</i>	85
2.2.5. <i>Crew Recovery Sub-Organization Implementation</i>	86
2.2.6. <i>JADE Implementation Examples</i>	88
3. RESULTS	91
4. CONCLUSIONS	97
LIMITATIONS AND FUTURE WORKS	101
REFERENCES	103
ANNEX A - BIBLIOGRAPHY	107
ANNEX B – PRELIMINARY ROLE MODEL	109
ANNEX C – PRELIMINARY INTERACTION MODEL	113
ANNEX D – ROLE MODEL	117
ANNEX E – INTERACTION MODEL	125
ANNEX F – SERVICE MODEL	131
INDEX	133

TABLE OF FIGURES

FIGURE 1 – PHASES AND INTERDEPENDENCES OF THE SCHEDULING PROBLEM	25
FIGURE 2 – DISRUPTION MANAGEMENT	28
FIGURE 3 – GAIA METHODOLOGY: A COMMENTED OVERVIEW.	37
FIGURE 4 - MASE PHASES	38
FIGURE 5 - JADE LOGICAL ARCHITECTURE	45
FIGURE 6	45
FIGURE 7 – ACTORS AND GOALS MAIN DIAGRAM FOR AN OPERATIONAL BASE	47
FIGURE 8 – DIAGRAM OF THE ENVIRONMENT MODEL.....	53
FIGURE 9 - ENVIRONMENT-PRELIMINARY ROLES DIAGRAM	57
FIGURE 10 – ENVIRONMENT – PRELIMINARY ROLES AND INTERACTION DIAGRAM.....	60
FIGURE 11 – FORCES INVOLVED IN THE IDENTIFICATION OF THE ORGANIZATIONAL STRUCTURE	65
FIGURE 12 - ORGANIZATION STRUCTURE WITH ENVIRONMENT MODEL	69
FIGURE 13 - ORGANIZATION STRUCTURE FOR CREW RECOVERY IN UML 2.0	70
FIGURE 14 - ORGANIZATION STRUCTURE FOR AIRCRAFT AND PASSENGER RECOVERY IN UML 2.0.....	70
FIGURE 15 - UML 2.0 INTERACTION DIAGRAM FOR INFORMCREWEVENTS PROTOCOL.....	74
FIGURE 16 - ENVIRONMENT - ROLE AND INTERACTIONS DIAGRAM.....	74
FIGURE 17 - UML 2.0 REPRESENTATION FOR ROLE AND INTERACTION MODEL OF CREW RECOVERY.....	75
FIGURE 18 - UML 2.0 REPRESENTATION FOR ROLE AND INTERACTION MODEL OF AIRCRAFT AND PAX RECOVERY	75
FIGURE 19 - UML 2.0 REPRESENTATION OF THE AGENT MODEL.....	79
FIGURE 20 - SERVICE MODEL (PARTIAL) FOR AGENT CLASS OpMONITOR	80
FIGURE 21 - CREW RECOVERY ARCHITECTURE.....	86
FIGURE 22 CREW EVENT	86
FIGURE 23 CREW SOLUTION LIST.....	86
FIGURE 24 CREW SOLUTION.....	86
FIGURE 25 - CREWEVENT CONCEPT IMPLEMENTATION CODE.....	88
FIGURE 26 - OpMONITOR AGENT CLASS IMPLEMENTATION.....	89
FIGURE 27 - MONITORCREWEVENTS SERVICE IMPLEMENTATION	90

TABLE OF TABLES

TABLE 1 – ACTORS, GOALS AND DEPENDENCIES.....	47
TABLE 2 – SUB-ORGANIZATIONS IN THE MULTI-AGENT SYSTEM.....	51
TABLE 3 – IDENTIFICATION OF RESOURCES AND ACTIVE COMPONENTS.....	53
TABLE 4 – ACTIONS PERFORMED ON THE RESOURCES AND RELATED PLANS OF THE CREW RECOVERY... 54	54
TABLE 5 – ACTIONS PERFORMED ON THE RESOURCES AND RELATED PLANS OF THE AIRCRAFT RECOVERY	54
TABLE 6 – ACTIONS PERFORMED ON THE RESOURCES AND RELATED PLANS OF THE PASSENGER RECOVERY.....	55
TABLE 7 – ROSTERCREWMONITOR PRELIMINARY ROLE.....	59
TABLE 8 – PRELIMINARY PROTOCOL DEFINITION FOR INFORMCREWEVENTS.....	61
TABLE 9 – LIVENESS (RELATIONS) ORGANIZATIONAL RULES.....	63
TABLE 10 – SAFETY (CONSTRAINTS) ORGANIZATIONAL RULES.....	63
TABLE 11 – SUMMARY OF TOPOLOGIES AND CONTROL REGIMES USED.....	67
TABLE 12 - ORGANIZATION STRUCTURE FOR CREW RECOVERY SUB-ORGANIZATION (FORMAL NOTATION)	67
TABLE 13 - ORGANIZATION STRUCTURE FOR AIRCRAFT RECOVERY SUB-ORGANIZATION (FORMAL NOTATION).....	68
TABLE 14 - ORGANIZATION STRUCTURE FOR PASSENGER RECOVERY SUB-ORGANIZATION (FORMAL NOTATION).....	68
TABLE 15 – LEGEND OF ORGANIZATION STRUCTURE DIAGRAM.....	69
TABLE 16 - ROSTERCREWMONITOR (RCM) ROLE.....	73
TABLE 17 – PROTOCOL DEFINITION FOR INFORMSCREWEVENTS.....	73
TABLE 18 - AGENT MODEL.....	77
TABLE 19 – SERVICE MODEL FOR AGENT CLASS OpMONITOR.....	78
TABLE 20 – CONCEPTS AND ACTIONS TO BE REPRESENTED AS CLASSES.....	82
TABLE 21 – NAMES TO BE USED IN IMPLEMENTATION.....	83
TABLE 22 – MAPPING OF JADE BEHAVIORS AND PROTOCOLS TO BE USED TO IMPLEMENT SERVICES.....	84
TABLE 23 – PERFORMATIVES TO BE USED FOR EACH INTERACTION PROTOCOL.....	85
TABLE 24 - CREW BEST PROPOSAL CHOICE ALGORITHM.....	87
TABLE 25 - CREW BEST PROPOSAL ALGORITHM COMPUTED VALUES.....	87
TABLE 26 - DESCRIPTION OF THE INFORMATION COLLECTED FOR EACH EVENT.....	91
TABLE 27 - CREW EVENT DATA USED IN THIS SCENARIO.....	92
TABLE 28 - CREW EVENT DATA DISTRIBUTION.....	92
TABLE 29 - CREW SOLUTION FOUND FOR EACH CREW EVENT USING METHOD 1.....	93
TABLE 30 - CREW SOLUTION FOUND FOR EACH CREW EVENT USING METHOD 2.....	94
TABLE 31 - METHOD 1 AND METHOD 2 COMPARED.....	95
TABLE 32 – PRELIMINARY PROTOCOL DEFINITION FOR REPORTCREWEVENTSTATUS.....	113
TABLE 33 – PRELIMINARY PROTOCOL DEFINITION FOR APPLYCREWSOLUTION.....	113
TABLE 34 – PRELIMINARY PROTOCOL DEFINITION FOR REPORTCREWSOLUTIONSTATUS.....	113
TABLE 35 – PRELIMINARY PROTOCOL DEFINITION FOR INFORMACEVENTS.....	114
TABLE 36 - PRELIMINARY PROTOCOL DEFINITION FOR REPORTACEVENTSTATUS.....	114
TABLE 37 - PRELIMINARY PROTOCOL DEFINITION FOR APPLYAIRCRAFTSOLUTION.....	114
TABLE 38 - PRELIMINARY PROTOCOL DEFINITION FOR REPORTAIRCRAFTSOLUTIONSTATUS.....	115
TABLE 39 - PRELIMINARY PROTOCOL DEFINITION FOR INFORMPAXEVENTS.....	115
TABLE 40 - PRELIMINARY PROTOCOL DEFINITION FOR REPORT PAXREQUESTSTATUS.....	115
TABLE 41 - PRELIMINARY PROTOCOL DEFINITION FOR APPLYPAXSOLUTION.....	116
TABLE 42 - PRELIMINARY PROTOCOL DEFINITION FOR REPORTPAXSOLUTIONSTATUS.....	116
TABLE 43 – CREWFIND (CF) ROLE.....	117
TABLE 44 - TAPCRHEURISTIC (TAPCR) ROLE.....	117
TABLE 45 - OTHERCRALGORITHM (OTHERCR) ROLE.....	118
TABLE 46 – OPERATIONSSCHEDULEMANAGER (OSM) ROLE.....	118
TABLE 47 – CREWASSIGN (CA) ROLE.....	119
TABLE 48– ROSTERAIRCRAFTMONITOR (RAM) ROLE.....	119
TABLE 49 - AIRCRAFTFIND (AF) ROLE.....	120
TABLE 50 - TAPARHEURISTIC (TAPAR) ROLE.....	120
TABLE 51 - OTHERARALGORITHM (OTHERAR) ROLE.....	121
TABLE 52– OPERATIONALCONTROLSUPERVISOR (OCS) ROLE.....	121
TABLE 53 - AIRCRAFTASSIGN (AA) ROLE.....	122

TABLE 54 - PAXMONITOR (PM) ROLE	122
TABLE 55 - PAXFIND (PF) ROLE	123
TABLE 56 - PAXAPPLY (PA) ROLE	123
TABLE 57 – PROTOCOL DEFINITION FOR REPORTCREWEVENTSTATUS	125
TABLE 58 – PROTOCOL DEFINITION FOR SENDSCREWSOLUTION	125
TABLE 59 – PROTOCOL DEFINITION FOR REQUESTSCREWSOLUTION	126
TABLE 60 – PROTOCOL DEFINITION FOR APPLYCREWSOLUTION	126
TABLE 61 – PROTOCOL DEFINITION FOR REPORTCREWSOLUTIONSTATUS	126
TABLE 62 – PROTOCOL DEFINITION FOR INFORMSACEVENT	127
TABLE 63 - PROTOCOL DEFINITION FOR REPORTACEVENTSTATUS	127
TABLE 64 – PROTOCOL DEFINITION FOR SENDSACSOLUTION	127
TABLE 65 – PROTOCOL DEFINITION FOR REQUESTSACSOLUTION	128
TABLE 66 - PROTOCOL DEFINITION FOR APPLYACSOLUTION	128
TABLE 67 - PROTOCOL DEFINITION FOR REPORTACSOLUTIONSTATUS	128
TABLE 68 - PROTOCOL DEFINITION FOR INFORMSPAXEVENT	129
TABLE 69 - PROTOCOL DEFINITION FOR REPORTPAXEVENTSTATUS	129
TABLE 70 – PROTOCOL DEFINITION FOR SENDSPAXSOLUTION	129
TABLE 71 - PROTOCOL DEFINITION FOR APPLYPAXSOLUTION	130
TABLE 72 - PROTOCOL DEFINITION FOR REPORTPAXSOLUTIONSTATUS	130
TABLE 73 – SERVICE MODEL FOR AGENT CLASS OpASSIGN	131
TABLE 74 – SERVICE MODEL FOR AGENT CLASS OpMANAGER	131
TABLE 75 – SERVICE MODEL FOR AGENT CLASS OpCRFIND	131
TABLE 76 – SERVICE MODEL FOR AGENT CLASS OpARFIND	132
TABLE 77 – SERVICE MODEL FOR AGENT CLASS OpPXFIND	132
TABLE 78 – SERVICE MODEL FOR AGENT CLASS OpTAPCRH	132
TABLE 79 – SERVICE MODEL FOR AGENT CLASS OpOTHERCRA	132
TABLE 80 – SERVICE MODEL FOR AGENT OpTAPARH	132
TABLE 81 – SERVICE MODEL FOR AGENT CLASS OpOTHERARH	132

ABBREVIATIONS AND SYMBOLS

AOCC	Airline Operations Control Centre
ASP	Airline Scheduling Problem
CSM	Crew Schedule Manager
DMAS	Distributed Multi-Agent System
MAS	Multi-Agent Systems
OC	Operations Control
OCS	Operations Control Supervisor
ORP	Operations Recovery Problem
VPN	Virtual Private Network

GLOSSARY

Aircraft Events	A flight delay (due to weather conditions, for example), an aircraft malfunction or other situation that can affect the schedule of the aircraft.
Aircraft Recovery	The process of solving (recovering) from problems related with aircrafts and triggered by aircrafts events.
Block Time	The period of time since the aircraft starts to move from the departure point until it stops in the destination point.
Cabin Crew Members	The crew members that provides on-board service to passengers. They are also responsible for the passenger security.
Cockpit Crew Members	The technical crew members responsible for piloting the aircraft.
Consecutive Critical Periods	Two consecutive periods of work between 02 and 06 am.
Crew Check-in	Action that represents the moment when the crew member reports for duty.
Crew Check-out	Action that represents the moment when the crew member finishes the duty.
Crew Events	A crew member that does not report for duty, a crew member delayed or other situation that can affect the company schedule.
Crew Pairing	A set of flights, starting and ending at home base of the crew members, that can have several days of duration.
Crew Rank	The rank of the crew member, for example, captain, first office, chief purser, etc.
Crew Recovery	The process of solving (recovering) from problems related with crew members and triggered by crew events.
Crew Rostering	The process of assign activities and pairings to crew members.
Crew Tracker Controllers	A person that works in the Crew Tracking Control that monitors and changes the roster (schedule) of a crew member.
Crew Tracker Managers	A person that works in the Crew Tracking Control that manages and takes decision regarding the roster (schedule) of a crew member.
Days Off	Days without any work or activity. Usually two days in each week.
Duty Period	The period from the start until the end of a pairing or other work activity of a crew member.
Duty Time	The period of time from the start until the end of a duty activity of a crew member.
Effectiveness	The degree of success in finding a solution meaning that, for the same event, two or more processes will find a solution when one solution is available.
Extra-Crew	A crew member that belongs to the crew of a flight without performing any duties on board. Usually used to position crew members from one airport to another.
Flight Legs	The flight from one airport to another. For example, flight from Lisbon to Caracas, might have to flight legs: Lisbon-Funchal and Funchal-Caracas.
Home Base	The operational base that the crew member belongs to.
Hub-and-spoke network structure	A system of air transportation in which local airports offers air transportation to a central airport where long-distance flights are available.
Operational Base	A company base with flights, crew members and aircrafts and with a schedule of flights that start and ends at the base.
Operations Controller	A person that works in the Operations Control that monitors and changes the roster (schedule) of an aircraft and/or of a flight.
Operations Managers	A person that works in the operations Control that manages and takes decision regarding the roster (schedule) of an aircraft and/or a flight.
Pairings	A set of flight legs, starting and ending at the same operational base.
Passenger Events	Any situation that can affect the passengers of a flight.
Passenger Recovery	The process of solving (recovering) from problems related with passengers and triggered by passenger events.
Perdiem Value	A value that is given to crew members, for each day of work, to pay for food or others expenses when they are out.
Published Schedule	The official schedule of the company regarding crew members and/or aircrafts. Defines the company operation.
Rest Time	The time, after a pairing or other duty, for a crew member to rest.
Roster	The schedule of a crew member.
Rotation of the Aircraft	The time between the arrival at a airport and the departure at the same airport.
Stand by Crew Members	Crew members that are scheduled so that they can be used to replace another crew in a flight.
Stand by Duties	Periods of time that a stand by crew member has to be waiting for a call case the company needs him to replace another crew member.
Stand by Roster	The set of individual stand by crew members roster. Means all the group of

	crew members that are schedule to be used when necessary to replace others.
Tail Numbers	The license number of the aircraft.

1. Introduction

1.1. Overview

One of the most important areas in an airline company is the Operations Control (OC). Through operations control mechanisms the airline company monitors all the flights checking if they follow the schedule that was previously defined by other areas of the company. The Airline Operations Control Centre (AOCC) has the responsibility to manage the airline operation during a specific time window that varies according to each company. The AOCC size and composition varies according to the size of the airline company but it is common to include teams specialized in Crew Scheduling, Aircraft Scheduling and Aircraft Maintenance, among others. Unfortunately, some problems arise during this phase (1). Those problems are related with crew members (for example, a crew member that did not report for duty), aircrafts (for example, a malfunction or a delay due to bad weather) and passengers (usually a consequence of the other two). The actions towards the solution of these problems are usually known as Disruption Management (2) or Operations Recovery. Due to the personal experience of the author as well as from the observation of how a real AOCC works, we found **(and this is the seminal hypothesis we put at this work starting point)** that the paradigms of Agents and of Multi-Agent Systems (MAS) as well as the advantages of such a system could be suitable to represent the AOCC and the several roles and functions that are performed in the AOCC. From (3) and (4) we point out the following characteristics/advantages that made us to propose the development of a Distributed Multi-Agent System (DMAS):

- ✓ A highly dynamic and complex environment. Systems with flexible autonomous actions are appropriate. MAS models problems in terms of autonomous interacting component-agents, which is a more natural way of representing task allocation, team planning, user preferences and so on.
- ✓ The organization environment (AOCC) is naturally modeled as a society of agents that cooperate with each other to solve complex problems. Some of the agents are “intelligent interfaces” that cooperate with the user to work on some problem.
- ✓ The distribution of data, control or expertise. Usually the AOCC controls more than one operational base, each one with resources (crew members and aircrafts) and, sometimes, with a local supervisor and local specialists in solving the problems.
- ✓ The distribution of computational resources. With a MAS we can distribute the computational resources and capabilities across a network of interconnected agents avoiding the problems associated with centralized systems.
- ✓ Legacy systems. The AOCC uses information that exists in obsolete but functionally systems. We can wrap the legacy components, providing them with “agent layer” functionality, enabling them to interact with other software components.
- ✓ A MAS efficiently retrieves, filters, and globally coordinates information from sources that are spatially distributed.
- ✓ A MAS provides solutions in situations where expertise is spatially and temporally distributed.

- ✓ A MAS is extensible, scalable, robust, maintainable, flexible and promotes reuse. All characteristics very important in systems of this dimension and complexity.

In this thesis we proposed the development of a Distributed Multi-Agent Systems (DMAS) that represents the AOCC. As the result of our observations we propose to include intelligent features that allow finding the best solution to the majority of the problems that arise during OC. The DMAS is also able to deal with different operational bases, including their own resources, solving the local problems and also contributing to the solution of problems in other bases.

We know that our proposal requires a huge effort in dedication and time and that we will not be able to do everything in this thesis. We believe that this work creates the necessary foundation and test workbench for future improvements in this realistic domain. The next sections are as follows:

- ✓ Section 1.2. Airline Scheduling Problem (ASP). Essential to understand how an airline company works before the Operations Control phase.
- ✓ Section 1.3. Operations Recovery Problem (ORP). The focus of this thesis.
- ✓ Section 1.4. Observations. The observations we made from the real AOCC of TAP Portugal (5) that motivated us to do this work.
- ✓ Section 1.5. Hypothesis and Predictions. Based on the observations we made some hypothesis and predictions that we try to prove (unfortunately not all of them) during this work.
- ✓ Section 1.6. Expected Results. Here we declare what results do we expect before starting our research.
- ✓ Section 1.7. State of the Art regarding Operations Recovery. Related work, done by other researchers, regarding this problem.
- ✓ Section 1.8. State of the Art regarding Agent-Oriented Methodologies. To be able to analyze, design and implement such a system we need to use methodologies and tools appropriate to agents and multi-agent systems. In this section we present the state of the art regarding this subject.
- ✓ Section 2. Methods Used. This section presents the work we have done regarding the design of our proposed system. It is divided in the following sub-sections:
 - 2.1. Vision and Scope: The vision and scope of our work is defined here.
 - 2.2. Methodology, tools and techniques: The methodology and tools used for analysis, design and implementation are introduced here.
 - 2.3. Requirements and Specifications: The early requirements analysis, analysis, architectural design, detailed design and implementation of our system are presented here.
- ✓ Section 3. Results. A comparison between the method we propose for solving crew operations recovery problem and the current method in TAP Portugal is presented here.
- ✓ Section 4. Conclusions. The discussion and conclusions of our work.

1.2. The Airline Scheduling Problem

According to (2) the interdependences of the several phases in the airline scheduling problem are illustrated as in Figure 1.

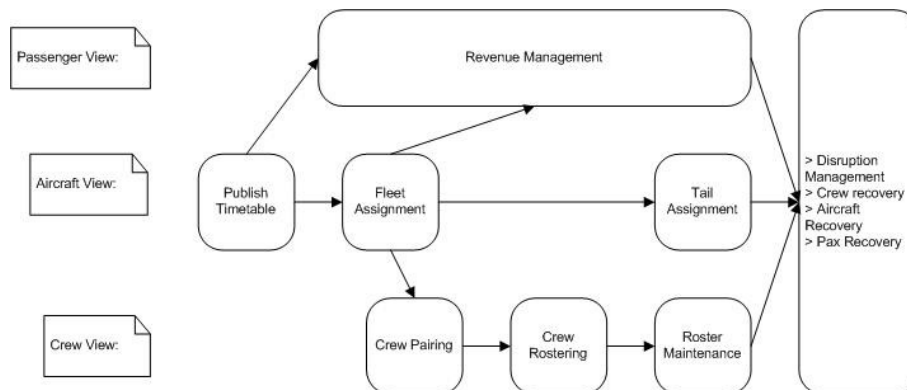


Figure 1 – Phases and interdependences of the scheduling problem

There are three views in the process that corresponds to three different areas of management. During the planning process they are seen as separate entities and can be scheduled and optimized more or less independently. Having the timetable and the fleet assignment, the planning process runs in parallel. A brief description of each phase follows:

Publish Timetable

The commercial flight schedule, that is, the schedule of flights that can be “sold” to passengers.

Fleet Assignment

The allocation of aircraft type (fleets) to the schedule flights.

After these two phases the planning process runs in parallel. The *crew scheduling problem* is usually solved in two phases: crew pairing and crew rostering.

Crew Pairing

In this phase, anonymous pairings (trips), starting and ending at home base, are constructed. The pairings must face all crew positions to be covered in the flights defined by the time table. Legal rules are applied at this time (for example, maximum duration of each flight duty period, minimum rest time between two flight duty periods, maximum consecutive critical periods, etc.) as well as some company rules (for example, minimum time for the rotation of the aircraft, maximum number of flight legs in each flight duty period, maximum number of duration days for each pairing, etc.).

Crew Rostering

This phase consists of assigning the pairings and other activities (for example, stand by duties, training, and reserves) and applying legal rules (for example, days off, vacation, and limits on duty/block time) to individual crew members. For a more detailed list please consult (1).

The crew scheduling (crew pairing + crew rostering) must be completed a few weeks before the day corresponding to the start of the schedule and the result of this phase is a personalized roster for each crew member, usually for a one month period.

Roster Maintenance

Later changes like commercial flights changes and crew availability are handled in this phase.

Tail Assignment

This phase consists of assigning the actual aircrafts (tail numbers) to flights and, consequently, the routing of aircrafts individuals. This is typically done a few days before day of operations. In TAP Portugal (5), for example, the personal roster is already published with the aircraft tail numbers. However, this assignment is not final and it is dependent of the operations control.

Revenue Management

Corresponds to the adjustment of prices and seat availability according to the market and is carried out during the entire period from the publication of the flight timetable to the day of operations.

Operations Control

Or Disruption Management as called in (2) corresponds to the monitoring of the schedule execution, trying to solve all the problems related with crew members, aircrafts and passengers. More information regarding this phase is given in the next section.

From the above information it is possible to see that the Airline Scheduling Problem is composed of several problems. None of these problems will be addressed in our work. However, for a better understating of our work we think that this information is helpful.

1.3. The Operations Recovery Problem

The Airline Operations Control Center (AOCC) is the entity that most airlines have to perform the tasks that are necessary to coordinate the schedule execution. The AOCC monitors the progress of operations, detects eventual problems and take actions in response to the unexpected events. The AOCC includes representatives of several key airline functions, that, working together ensures a smooth schedule execution. There are some differences in the composition of the AOCC, especially when comparing European and North American companies (2). The most common support roles (2) in the AOCC are the following:

- ✓ Aircraft control: This role, besides managing the resource aircraft, is the central coordination role in operations control. In Europe it is divided in long and short haul flights.
- ✓ Crew tracking: This role is responsible for assuring that every flight has the necessary crew members to operate. They monitor the report for duty of crew members (crew check-in) and act accordingly when someone does not report for duty. They also change crew pairings in case of flight delays and cancellations. In most airlines this role is divided into cockpit and cabin crew.
- ✓ Aircraft engineering (maintenance): Aircraft engineering is responsible for unplanned service and maintenance of the aircraft as well as the short term maintenance scheduling. Changes to aircraft rotations may impact on short term maintenance because the maintenance cannot be done at all stations.
- ✓ Customer service: The decisions taken at the AOCC might affect passengers. This role is responsible to ensure passenger inconvenience is taken into consideration in these decisions. Delays and cancellations will affect passengers who need to be informed and in some cases rebooked or provided with meals or accommodation. The customer service is provided at the gates and in customer service center, which are not part of the AOCC.

Every change in the schedule, for example, a flight delay or cancellation, aircraft fleet change, crew members that do not report for duty, new flights scheduled, etc., that happens in this phase must be feasible for crew as well as for aircraft and should minimize the passenger inconvenience. The AOCC tries to solve these problems with the minimum impact in the airline schedule, with the minimum cost and, at the same time, satisfying all the required safety rules. For this reason, the successful operation of an airline depends on the coordination of the actions of all supporting functions. Every AOCC has a person that assumes the responsibility of the overall coordination of operations, ensuring that all groups acts as one team.

It is in the AOCC that the Operations Recovery/Disruption Management Process takes part. A high-level view of this process as found in (2) is presented in Figure 2.

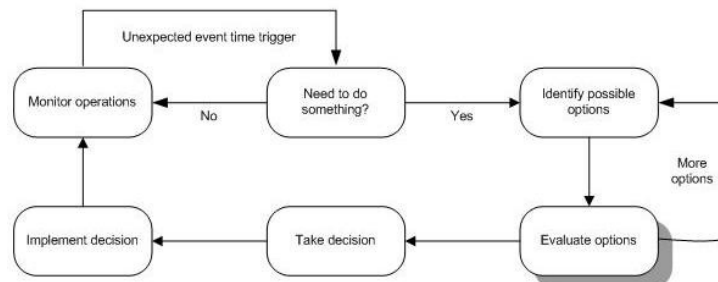


Figure 2 – Disruption management

The airline constantly monitors its operations. When an event happens, for example, a crew member that did not report for duty or a flight delay due to weather conditions, it is necessary to evaluate the need to do something. Some of the events are minor, like small flight delays and, in this case, no actions are needed. If an action is needed it is necessary to identify the possible options and evaluate them. The evaluation will consider the passenger, aircraft and crew perspective. These evaluations may result in proposed changes to the option. With all the information collect it is possible to take a decision. That decision can be implemented immediately or postponed until a more adequate time.

During this phase, we call *Crew Recovery* to the process of solving crew related problems, *Aircraft Recovery* to the process of solving aircraft related problems and *Passenger Recovery* to the process of solving passenger problems.

1.4. Observations

For our work we have observed the AOCC of TAP Portugal (5) and arrived at the following observations:

Observation 1

TAP Portugal has three operational bases localized in Funchal, Lisboa and Porto. Funchal base has cabin crew members (although not all ranks) based there but no cockpit crew members. There are flights that start and end at this base and, because of that, some aircrafts are based in Funchal too. Lisboa base is the largest one and has cabin and cockpit crew members based there as well as the majority of the aircrafts. Porto base has cabin and cockpit crew members and has more flights starting and ending there then Funchal. Some aircrafts are also based in Porto.

Observation 2

TAP AOCC organization is a multilevel hierarchy. They have an Operations Control Supervisor (main authority regarding the OC and also regarding aircraft recovery and passenger recovery) and a Crew Schedule Manager (main authority regarding crew schedule changes). Despite the existence of a main authority, in practice, these two roles are at the same level. Level one of the hierarchy is composed by these two roles. The second level is composed by leaders of the teams specialized in each type of problems. Finally, the third level is composed by the specialized team elements. We also observed that there are a mixed of cooperation (between elements of the third level and between members of the first level) and authoritative control (between different levels). The monitoring of the operations is done by three different roles: one monitoring crew events, other monitoring aircraft events and, finally, other monitoring passenger events. When an event is detected (flight delay, crew delay, passenger boarding problems) the person that detected it informs the other teams. Each team evaluates the situation and, if it is the case, tries to find a solution regarding their domain of expertise. When necessary the teams share information cooperating with the other teams. The final decision is taken by the Operations Control Supervisor (OCS) but, if it is necessary to make any change in the crew schedule, the OCS and the Crew Schedule Manager (CSM) need to agree before the final decision. The implementation of the decision is made by each team according to their domain.

Observation 3

The information system that supports the AOCC is centralized in Lisboa. This includes the databases even the ones with resources specific to each operational base. There are some workstations in each base for the crew members to check-in (report for duty) and to check-out (at the end of the duty). The workstations for the users that perform tasks in the AOCC are in Lisboa. Other information systems, like human resources and payroll, are also localized in Lisboa. It is possible to access the information on these information systems from other places, through a dedicated communications line or VPN (Virtual Private Network).

Observation 4

The information system available for the users in the AOCC that have to find solutions to the problems, only has operational information, that is, information that is

necessary for the operation to be performed (for example, aircraft status and maintenance, crew data related with licenses, qualifications and roster). They can have access to airport information or weather information, for example, but not in an integrated way. They need to use different workstations for that.

For example, they do not have access regarding payroll or to the costs related with extra-crew trips (crew members that travel from one base to another without performing a specific tasks in the flight).

Observation 5

During the process of operations recovery the users apply knowledge that was transmitted by older colleagues and/or from the training they received, but also knowledge that results from their own experience. They also know, by their own experience, the preferences of some of the crew members and try to use that information in solving crew related problems. Something similar happens for the other types of problems. Although this knowledge is used in future situations as a reaction to similar problems, it is not used in a pro-active way, trying to minimize or eliminate the problems by anticipating solutions.

In the next section we introduce the hypothesis, based on the above observations, which we believe are relevant for what we want to achieve.

1.5. Hypothesis and Predictions

Hypothesis 1

Based on the first observation we hypothesize that the main objective of an airline operation (that is, flights always on time) will be much easier to achieve (that is, less flights delayed) if we take advantage of the fact that each operational base has specific resources (crews and aircrafts). We predict that if we solve the problems first with local resources and then with non-local resources, the solutions to the eventual problems will be much faster to find. In some cases, the non-local solutions might be the only solution available.

Hypothesis 2

Based on the third and fourth observation we hypothesize that the objective of solving the operations recovery problems with the less cost as possible, will be much easier to achieve if we consider information in the decision process that is not available in the information system used in the AOCC, related with various costs and crew payroll. Regarding crew recovery problems, we predict that if we take into account payroll information like *hour salary* and *perdiem value* of each crew rank, and costs related with hotels and *extra-crew travel*, the solution will be less expensive. The same principle can be applied to aircraft recovery and passenger recovery, using costs related with that domain.

Hypothesis 3

Based on the second observation we hypothesize that the Agent and Multi-Agent Systems paradigm is appropriate for the development of a system that represents the AOCC organization. We predict that if we build a system according to the MAS paradigm, the organization will benefit from the advantages of such a system (see chapter 1.1 and (3),(4)) and the main objective of an airline operation (see hypothesis one) will be much easier to achieve. At the same time the MAS will allow faster and better decisions (according to the criteria defined by the company, usually related with costs and compliance with rules) without compromising the importance of the human supervisor in the process.

Hypothesis 4

Based on the fourth observation we hypothesize that the use of different algorithms to solve the same problem (in crew and aircraft recovery) will improve the achievement of the main objective of the crew and aircraft recovery process (that is, to always find the better solution regarding “ensuring every flight has a crew” and “ensuring that all flights are on time”, respectively), contributing to a more robust system. We predict that using different algorithms (genetic algorithms, heuristic, etc.) in comparison with using always the same algorithm, to solve the same problem, will permit to always find the *best* solution (according to the criteria defined by the company) and to always find *a* solution, especially taking into account the fact that we might benefit from solutions presented by other bases, as stated in the first hypothesis and, thus, increasing the robustness of the system.

Hypothesis 5

Based on the fifth observation we hypothesize that the implementation of a learning mechanism that will learn from the utilization of crew members (in comparison with the previous and published schedule and in characterizing specific situations) will

permit a better use of the resources (especially crew members) in future schedules. We predict that, for example, if we learn the real use of stand by crew members in each month and in specific situations, will allow adjusting the stand by roster in similar months or similar situations of future schedules, permitting to release crew members to be schedule to flights (crew members are one of the most expensive resources in an airline company).

Hypothesis 6

Based on the fifth observation we hypothesize that if we extend the learning mechanism to learn the profile of each crew member, regarding his/her preferences, will increase the level of satisfaction of them. We predict that applying the learned profile of each crew member in future schedules of corresponding months will produce a roster that will achieve the goals of the airline company and, at the same time, the satisfaction of the crew members. Increasing the level of satisfaction of a crew member will decrease the crew member's absence to work.

1.6. Expected Results

Regarding the first hypothesis we expect to obtain solutions faster, that is, in less time than the present process at TAP Portugal and a greater use of local resources in comparison to the use of non-local resources (that is, from a different operational base than the one where the event appear). We also expect to obtain an increase in cooperation from different operational bases, manifested by the use of resources from other operational bases when the local resources are not available. In a more formal way:

$$\begin{aligned} \text{Time}(\text{SolutionMAS}(x)) &\leq \text{Time}(\text{SolutionTAP}(x)) \\ \text{Use}(\text{LocalRes}(\text{SolutionMAS}(x))) &\geq \text{Use}(\text{LocalRes}(\text{SolutionTAP}(x))) \\ \text{Cooperation}(\text{SolutionMAS}(x)) &> \text{Cooperation}(\text{SolutionTAP}(x)) \end{aligned}$$

Regarding the second hypothesis we expect to obtain a considerable decrease in the costs of the solutions for the problems found when compared with the costs of the solutions found by the present process at TAP Portugal. In a limit and assuming that our proposed system and the current process in TAP always have the same effectiveness (that is, the same degree of success in finding a solution meaning that, for the same event either process will find a solution when one solution is available) the solution obtain by our system will not be more expensive than the solution found by the current process in TAP. In a more formal way:

$$\text{Cost}(\text{SolutionMAS}(x)) \leq \text{Cost}(\text{SolutionTAP}(x))$$

$$\begin{aligned} \text{Subject to:} \\ \text{Effectiveness}(\text{SolutionMAS}(x)) &= \text{Effectiveness}(\text{SolutionTAP}(x)) \end{aligned}$$

Regarding the third hypothesis we expect to obtain a design of a MAS that can incorporate the needs of the AOCC, performing automatically the more repetitive tasks (for example, monitoring of events), gathering faster and more complete information, allowing the human supervisors to take the final decision based on a more complete information, take advantage of the distributed resources and of the distributed architecture of the system, and, finally, allow the system to grow according the needs either in terms of more operational bases or in terms of new or improved search algorithms (scalability).

Regarding the fourth hypothesis we expect to obtain better solutions (according to the criteria defined by the company) than the solution obtained by the current process and to always find a solution (assuming that one solution exists) even when the current process does not find one solution to the same problem. We expect that the heterogeneity of the algorithms, specialized in different types of problems, will allow to find solutions especially for the non-trivial problems.

Regarding the fifth hypothesis and as the major goals, we expect to have more crew members available to be schedule to flights and less operational recovery problems. We expect that future crew rosters, after incorporating the knowledge gather during operations recovery, will not include the schedule of activities that originate operations problems (for example, short time for changing from one aircraft to another in the same duty period) avoiding that the same problems appears during the

operations control phase. We also expect to see that future stand by rosters are much more adequate (that is, includes only the necessary crew members) to the real use of the stand by crew members in corresponding periods.

Regarding the sixth hypothesis we expect to have an increase in the satisfaction level of the crew members and, consequently, less operational problems. We expect that the knowledge gather from the profile of each crew member be included in the generation of rosters of the corresponding periods.

1.7. State of the Art: Operations Recovery

Traditionally the Airline Scheduling Problem, including the Operations Recovery Problem, has been solved through Operations Research (OR) techniques. The paper (6) gives an overview of OR applications in the air transport industry.

The literature that exists related with this subject is usually divided according to the specific resource to be recovered. The most common are aircraft, crew and passengers. However, it is also possible to find papers related with more general approaches as well as related with integrated recovery approaches. We will present here the most recent published papers according to (7). We divided the papers in four areas: general approaches, aircraft recovery, passenger recovery and integrated recovery. For a more detailed explanation of the papers as well as for older papers related with each of these subjects, please consult (7).

General Approaches

In (2) the author's reports on the experiences obtained during the research and development of project DESCARTES (a large scale project supported by EU) on airline disruption management. The current (almost manual) mode of dealing with recovery is presented. They also present the results of the first prototype of a multiple resource decision support system.

Aircraft Recovery

The most recent paper considering the case of aircraft recovery is dated from 2002 (8). The proposed model addresses each aircraft type as a single problem. They formulate the problem as a Set Partitioning master problem and a route generating procedure. The goal is to minimize the cost of cancellation and retiming, and it is the responsibility of the controllers to define the parameters accordingly. To solve the master problem in due time, a heuristic is used to select only a subset of aircraft to be involved in the Set Partitioning problem. This approach results in running times between 6 and 16 seconds for 3 real-size problem instances. It is included in the paper a testing using SimAir (9) simulating 500 days of operations for three fleets ranging in size from 32 to 96 aircraft servicing 139-407 flights.

Crew Recovery

In (10) the flight crew recovery problem for an airline with a hub-and-spoke (a system of air transportation in which local airports offers air transportation to a central airport where long-distance flights are available) 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. Based on detailed information regarding the current plan and pool of problems, the recovery problem is solved in steps. Several means are used for recovery, including delaying, swapping, deadheading (extra-crew) and the use of stand by crew. The proposed model is an assignment model with side constraints. Due to the stepwise approach, the proposed solution is sub-optimal. Results are presented for a situation from a US airline with 18 problems.

Integrated Recovery

It is uncommon to find literature dedicated specifically to the passenger recovery problem. We believe the main reason for this is the fact that the passenger problems can be minimized if we solve the aircraft and crew problems. However, we would like to point out a recent paper (11) that, although presenting an integrated recovery

approach, has a strong emphasis on reducing passenger arrival delays. This paper presents two models that considers aircraft and crew recovery and through the objective function focuses on passenger recovery. These are based on the flight schedule network. Although crew is incorporated into the models they do not consider how to recover from disrupted crews. To test the models an AOCC simulator was developed, simulating domestic operations of a major US airline. It involves 302 aircrafts divided into 4 fleets, 74 airports and 3 hubs. Furthermore, 83869 passengers on 9925 different passengers' itineraries per day are used. Three different scenarios with different levels of disruption are presented. Execution times ranges from 201 to 5042 seconds. For all scenarios are generated solutions with reductions in passenger delays and disruptions.

Letovsky's Ph.D. thesis (12) is the first presentation of a truly integrated approach in the literature, although only parts of it are implemented. The thesis presents a linear mixed-integer mathematical problem that maximizes total profit to the airline while capturing availability of the three most important resources: aircraft, crew and passengers. The formulation has three parts corresponding to each of the resources, that is, crew assignment, aircraft routing and passenger flow. In a decomposition scheme these three parts are controlled by a master problem denoted the Schedule Recovery Model.

Finally, we would like to point out a tool called DART (Decision-Aided Rescheduling Tool) (13) that was developed to control the flight operations of IBERIA (the Spanish airline company). DART controls airline operations by gathering real time world-wide information about fleet and crew situation and providing decision support for handling incidents. It covers the daily execution of the ideal flight plan and is responsible for tracking and solving any irregularities that might arise during its execution. The authors claim that DART has been able to solve some difficult problems, proposing, in some cases, better solutions than those proposed by the re-scheduling experts. The paper does not present any comparative results.

In addition to the above literature the conferences of the AGIFORS organization (14) often feature presentations related with operations recovery. The contributions from these conferences are, at best, available in the form of presentation slides. As such, we did not consider them here.

1.8. State of the Art: Agent-Oriented Methodologies

Agent technology in the context of software engineering has received a lot of attention during the last few years. Agent technology has been very successful from the scientific point of view as a metaphor for decentralized computation. From the commercial point of view we start to see some real-world agents and multi-agent systems applications. For example, the Distributed Computing with the Digipede Network (15) that uses agents to make distributed computing a reality, and the Infomobility Services application from the IST IMAGE project (16).

Some agent-oriented software development methodologies, either general-purpose approaches or special-purpose approaches, have been proposed. Some examples are, Message/UML (17), Tropos (18), Prometheus (19), MaSE (20)(21), Passi (22), and Gaia (23). A brief summary of some of the general-purposes methodologies, based on (24), follows:

GAIA

GAIA has three phases: analysis, architectural design and detailed design. Figure 3 gives a commented overview of this methodology.

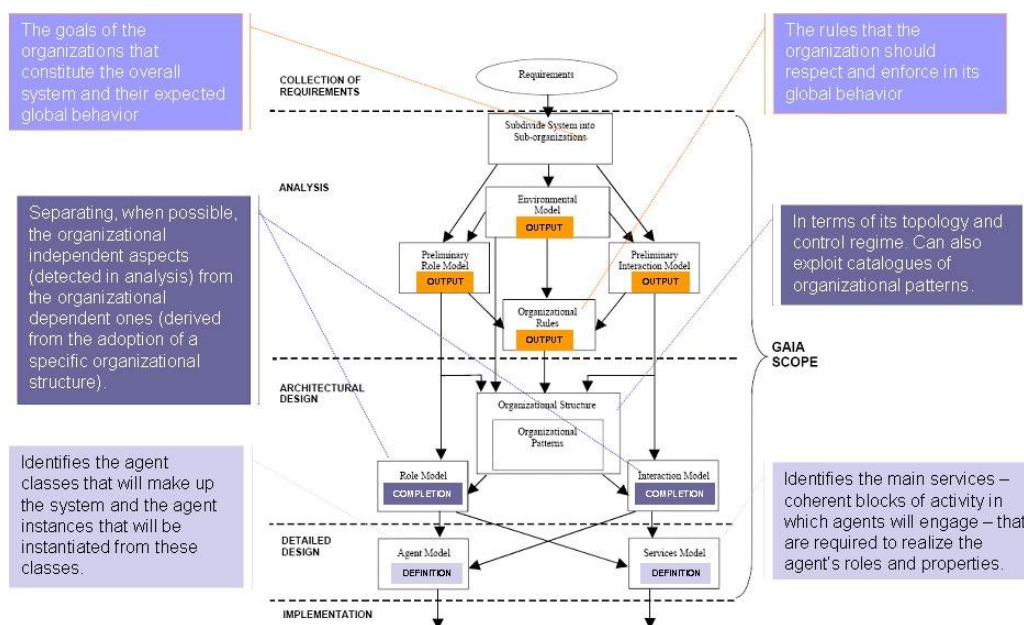


Figure 3 – GAIA Methodology: a commented overview.

In the analysis phase this methodology produces the following outputs: The environment model, the preliminary role model, the preliminary interaction model and the organizational rules. In the architectural design the role and interaction model are finished and in the detailed design the Agent Model and the Services Model are defined. GAIA does not deal directly with implementation issues. The results are a detailed but technologic-neutral specification.

TROPOS

Tropos is based on two key ideas. First, the notion of agent and related mentalistic notions, such as goals and plans, are used in all phases of software development, from early analysis down to the actual implementation. Second, Tropos covers the very early phases of requirements analysis, thus allowing for a deeper understanding of the

environment where the software-to-be will eventually operate. Tropos has four phases: early requirements analysis, late requirements analysis, architectural design and detailed design. In the early requirements analysis phase the intentions of the stockholders are modeled as goals that, through a goal-oriented analysis, eventually lead to the functional and non-functional requirements of the system-to-be. The late requirements analysis results in a requirements specification which describes all functional and non-functional requirements. The architectural design describes how system components work together. Tropos has defined organizational architectures styles for cooperative, dynamic and distributed applications (such as MAS) to guide the design of the system architecture. In the detailed design additional detail for each architectural component of the system are introduced. In particular, this phase determines how the goals assigned to each actor are fulfilled by agents in terms of design patterns. Detail design also includes the specification of agent communication and agent behavior. TROPOS has a tool that supports all these phases including the generation of JADE code “TAOM4E - Tool for agent oriented visual modeling for the Eclipse platform” (25).

MASE

The Multiagent Systems Engineering Methodology (MaSE) is a full-lifecycle methodology for analyzing, designing and developing heterogeneous MAS. MaSE views MAS as a further abstraction of the object-oriented paradigm where agents are specialized objects. Instead of simple objects, with methods that can be invoked by other objects, agents coordinate with each other via conversations and act proactively to accomplish individual and system-wide goals. MaSE has two phases: analysis and design (see Figure 4).

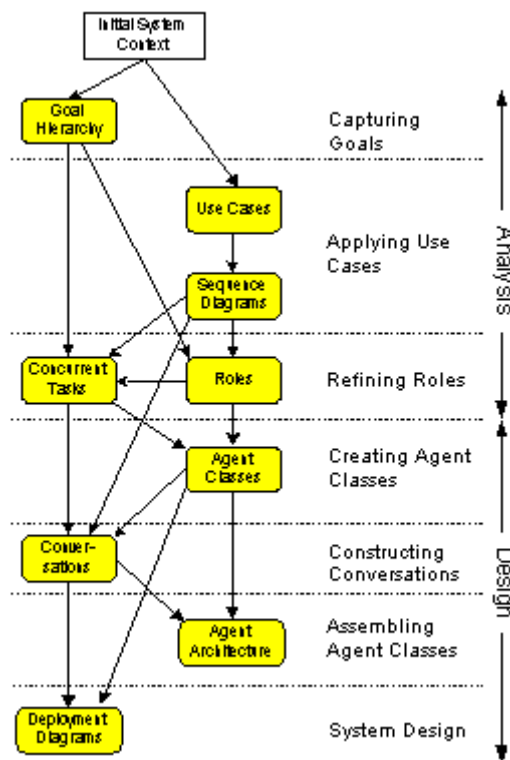


Figure 4 - MaSE phases

The analysis phase consists of three steps: capturing goals (through a goal hierarchy model), applying use cases (through use cases and sequence diagrams models), and refining roles (through concurrent tasks and role models). The design phase has four steps: creating agent classes (through agent classes diagrams), constructing conversations (through conversation diagrams), assembly agent classes (through agent architecture diagrams), and system design (through deployment diagrams). The methodology is iterative allowing the designer to move between steps and phases such that with each successive pass, additional detail is added and, eventually, a complete and consistent system design is produced. MaSE has a graphically based tool called agentTool (26), which fully supports each step of MaSE analysis and design. This tool also supports automatic verification of inter-agent communications, semi-automated design, and code generation for multiple MAS frameworks. MaSE and agentTool are both independent of any particular agent architecture, programming language or communication framework.

We have also seen some attempts to provide roadmaps e.g. (27) and tools e.g. (28), (29) for allowing analysis and design methodologies to be implemented using JADE (30). Recently, Gaia2JADE (31), has been proposed as a process in order to develop real-world MAS analyzed and designed using the Gaia methodology and implemented with the JADE framework. Gaia2JADE enhances the Software Process Engineering Metamodel proposed by the Object Management Group (32), adding the JADE development phase, proposing a process that covers the full software development cycle.

For more detailed information of the above methodologies and others that we did not mention here, we recommend the reading of (24).

2. Methods Used.

In this section we present all the steps performed to specify our proposed system. We begin by defining the vision and scope of our system and the methodology, tools and techniques used. We follow the methodology, presenting the requirements analysis, the system analysis, design and the implementation.

2.1. Vision and Scope

To achieve the expected results for the third hypothesis we have done an analysis and design for a Distributed MAS, applied to Airline Operations, with emphasis in the following:

- ✓ Monitoring events.
- ✓ Automate the resolution of the most trivial problems in Crew and Aircraft Recovery.
- ✓ Solve problems for each operational base and share available resources with other bases.
- ✓ Open to solutions from other airline companies.
- ✓ Learning preferences and new solutions.
- ✓ Robustness through redundancy (33).

The main problems that arise during operations control are the following (for a more detailed list of typical problems consult (1)):

Flight delays

Due to weather conditions, air traffic control restrictions, boarding problems, etc. This type of problems can lead to delays on the flights that depend on the arrival of the aircraft and/or in connection flights. Crew problems might arise.

Aircrafts malfunctions

This type of events can lead to problems similar to the previous one. Crew and aircraft problems might arise.

Crew members not reporting for duty

It is necessary to find an available crew member for replacement. This type of events can lead to flight delays in some conditions or, in the most complicated cases, to flight cancellation.

Crew member's delays in reporting for duty

It might be necessary to call the stand by crew. This type of events might lead to problems similar to the previous one.

Commercial changes to the flights

Changes in the flight schedule, new flights, cancellation of flights and so on. Usually leads to a crew scheduling (pairing and rostering) problem.

The final goals of our MAS where:

Problem anticipation

A successful and effective monitoring of events should be able to forecast possible problems and allow on time resolutions.

Detailed and correct information

The system should be able to give the necessary information for the users to understand how the solutions were achieved or, in some rare cases, allow the users to find a solution.

Apply the solution

The system should be able to apply automatically the best solution found or, in some special cases, the solution selected by the user. This might include the automatic creation of crew pairings and their assignment to crew members.

Crew member profile

The system should be able to learn the crew members bid preferences as well as the individual execution of the crew roster.

Use of the stand by crew members

The system should be able to learn the utilization of the stand by crew members during operations control. The knowledge that came up from this learning should be used in future crew scheduling. This will allow a better use of this rare and expensive resource.

Quick solution for trivial problems

Crew problems like the replacement of a crew member and/or flight delays that lead to crew problems, should be solved quickly and automatically.

From controllers to managers

The system should allow a reduction of the number of users that usually take care of operations control. The idea is to have Operations Managers/Crew Tracker Managers instead of Operations Controller and Crew Tracker Controllers. The system should do the control and the users should be the managers.

Robustness through redundancy

The system should be able to find solutions to the same problem using different algorithms.

As one can see this domain has very complex problems. To be able to achieve the goals we came up with the following list of features to be implemented:

- ✓ Monitoring crew members reporting for duty.
- ✓ Monitoring of flight departures and arrivals.
- ✓ Monitoring stand by crew members.
- ✓ Monitoring commercial changes to the flight schedule.
- ✓ Log absences communicated by crew members to the operations control.
- ✓ Implement several algorithms to be used in finding the solutions: Evolutionary Computing, Column Generation, Integer Programming, Heuristic Methods and others.

- ✓ Assign automatically the best crew to open positions in a pairing after choosing the best solution from the several proposed by the different algorithms.
- ✓ Create/change pairings resulting from commercial flight changes or other problems after choosing the best solution from the several proposed by the different algorithms.
- ✓ Log and learn the profile of each crew member related with bid preferences to be used in future crew scheduling.
- ✓ Learn the profile of each crew member from the individual execution of each crew member schedule and apply that profile in future crew scheduling.
- ✓ Learn from the use of the stand by crew members and apply that knowledge in future crew scheduling.
- ✓ Learn from the applied solutions and with that knowledge propose new solutions.

2.2. Methodology, Tools and Techniques

The main methodology that we used was GAIA (23). Following the suggestions of the authors of GAIA and because the collection of requirements are an input for this methodology, we have adopted a goal-driven early-requirements analysis (34), (18) for early-requirements analysis and UML 2.0, based on suggestions presented in (35), as the notation to use when appropriate.

To model and develop the MAS we use IBM Rational Software Architect (36) and, for the database, IBM Universal DB2 (37).

Finally we use Java Agent Development Framework (30) as the middleware platform for our system. JADE is a pure Java, middleware platform intended for the development of distributed multi-agent applications based on peer-to-peer communication. JADE includes Java classes to support the development of applications agents, and the “run-time” environment that provides the basic services for agents to execute. Figure 5 shows the logical architecture of JADE and Figure 6 the distributed architecture of a JADE agent platform.

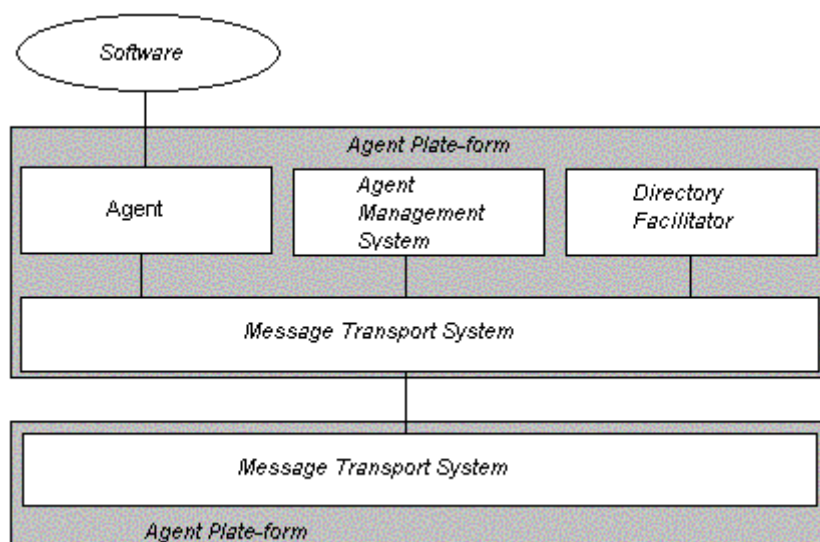


Figure 5 - JADE Logical architecture

Distributed architecture of a JADE Agent Platform

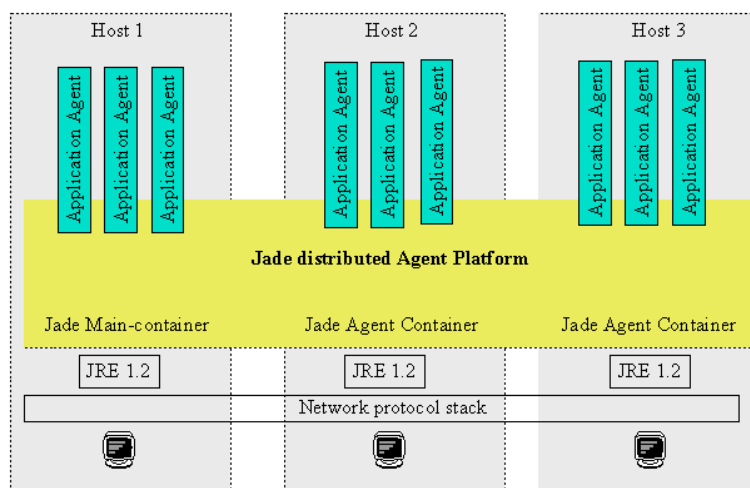


Figure 6

2.1. Requirements and Specifications

2.1.1. Early Requirements Analysis

The result of the early requirements analysis done through a goal-oriented modeling (18) is displayed in the diagram in Figure 7.

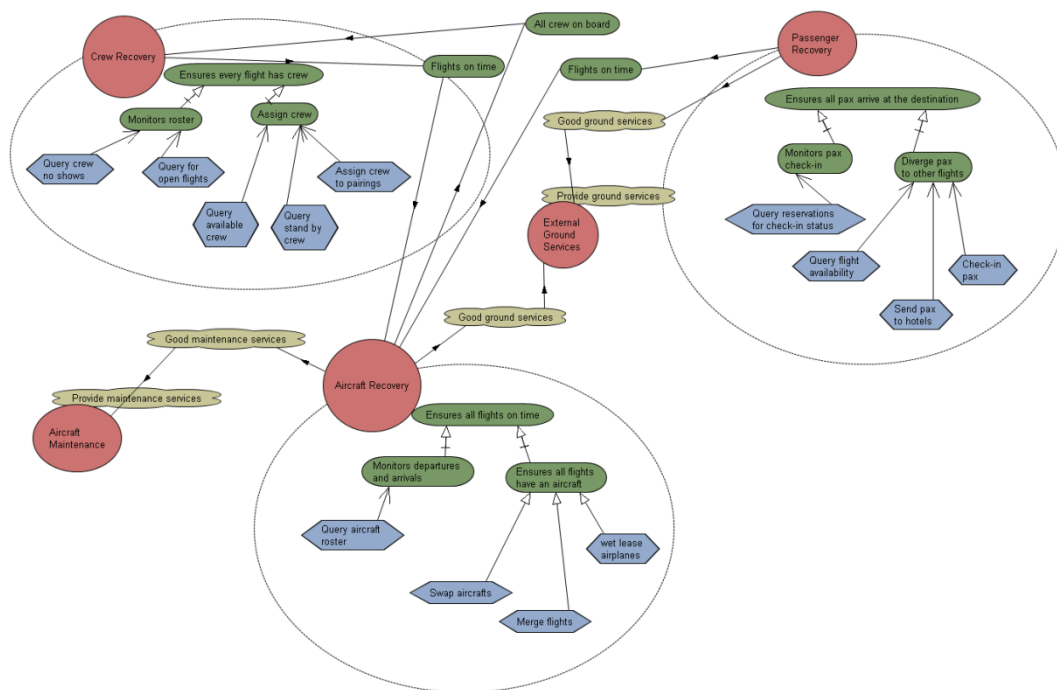


Figure 7 – Actors and goals main diagram for an operational base.

We point out that this diagram represents the requirement analysis for one operational base. An airline company can have more than one operational base. The requirements for the other operational bases will be the same.

From this diagram we can take the following list of actors and corresponding goals and soft-goals to be achieved, as well as goals and soft-goals dependencies from other actors. This information will also help to understand the roles of each one:

Actors	Goals	Soft-goals	Dependencies
Crew recovery	Ensures every flight has crew		Aircraft recovery <i>Flights on time</i> goal
Aircraft recovery	Ensures all flights on time		Crew recovery <i>All crew on board</i> goal; Aircraft maintenance <i>Good maintenance services</i> soft-goal; External ground services <i>Good ground services</i> soft-goal;
Passenger recovery	Ensures all passengers arrive at the destination		Aircraft recovery <i>Flights on time</i> ; External ground services <i>Good ground services</i> soft-goal;
Aircraft maintenance		Provide maintenance services	
External ground services		Provide ground services	

Table 1 – Actors, goals and dependencies.

From this analysis we also have conclusions regarding the way each actor should fulfill their objective. Most of the objectives can be modeled by AND/OR decomposition and by a MEAN-END analysis. The analysis for each of the actors involved is the following:

Crew Recovery Actor

The main goal of this actor is “*Ensures every flight has crew*”, meaning that before departure, it is necessary to guarantee that all flights have all crew members assigned according to the regulations. In this analysis we did not consider the change of existing crew pairings, either splitting or creating new ones. We assume that the environment already has the necessary pairings. To be able to achieve this goal it is necessary to do an “and decomposition”, meaning that it is necessary to achieve the sub-goal “*Monitors roster*” AND “*Assign crew*”. To fulfill the sub-goal “*Monitors roster*” all of the following plans are necessary to be executed:

- ✓ “*Query crew no shows*”, meaning that it is necessary to query one of the resources available in the environment to obtain the name of the crew members that did not report for duty. Those crew members will be replaced by others.
- ✓ “*Query for open flights*”, to query for flights with open crew positions, meaning that it is necessary to assign crew members to that flights.

To fulfill the sub-goal “*Assign crew*” it is necessary to execute all these three plans:

- ✓ “*Query available crew*”, obtain a list of crew members that are available to do the flight that has an open position. In this case, available means that the crew member does not have any kind of activity assigned, including a day off, and that, according to the regulations, can be assigned to the flight.
- ✓ “*Query stand by crew*”, obtain a list of crew members that have assigned a stand by activity and that, according to the regulations, can be assigned to the flight.
- ✓ “*Assign crew to pairings*”, finally, from the two lists obtained from the previous plans, choose the best crew member according to criteria defined by the company, and assign him to the flight.
- ✓ The execution of the above plans is a mean to achieve the mentioned goals.

Aircraft Recovery Actor

This actor has a similar objective but, in this case, the resources are aircrafts instead of crew members. The main goal of this actor is “*Ensures all flights on time*”. The idea is to avoid as much as possible the flight delays. To be able to achieve this goal an “and decomposition” has been applied, meaning that it is necessary to achieve the sub-goal “*Monitors departures and arrivals*” AND “*Ensures all flights have an aircraft*”.

To fulfill the first sub-goal it is necessary to execute the following plan:

- ✓ “*Query aircraft roster*”, obtain a list of delayed flights. This list should contain the departure and arrival delays and the aircrafts affected.

To fulfill the second sub-goal it is necessary to execute any of these three plans (in this case we have an OR decomposition):

- ✓ “*Swap aircrafts*”, the idea is to use another aircraft from another flight so that the delayed flights can departure on time. Of course that some conditions need to exist before it is possible to execute this plan. For example, the aircraft used must be assigned to a flight with a departure time later than the delayed flight. This will give time for the maintenance department to solve the problem with the aircraft and, hopefully, will be used for the following flight.
- ✓ “*Merge flights*”, the idea is to use the aircraft of another flight to do the delayed flight. For example, suppose that in Lisbon airport there is a flight to Paris departing at 15 pm and a flight to London departing at 15.15 pm. If the aircraft assigned to the Paris flight has a severe malfunction, it is possible to use the aircraft assigned to the London flight and merge the flights, resulting in a flight from Lisbon to Paris and then to London. Of course that there are a lot of things to take in consideration, like the fact that the passengers for London will arrive much later than expected and that the return flight will also be delayed.
- ✓ “*Wet lease airplanes*”, the idea is to lease an aircraft and crew to another airline company. In this case, the flight would start with a minimum delay and the maintenance services will have time to fix the aircraft malfunction.

Passenger recovery Actor

This actor has to “*Ensure all passengers arrive at the destination*”. The idea is to guarantee that the passengers will get to the destination, as soon as possible, after an expected long delay or in case of an overbooking situation. To achieve its goal, it is necessary to fulfill the following sub-goals:

- ✓ “*Monitors passengers check-in*” AND
- ✓ “*Diverge passengers to other flights*”.

To fulfill the “*Monitors passengers check-in*” sub-goal it is necessary to execute the following plan:

- ✓ “*Query reservations for check-in status*”, obtain a list of the check-in status of each reservation. This list will allow to identify the passengers that will not show for the flight and, in the case of an overbooking situation, will allow accommodating other passengers. This list will also identify all the passengers already checked-in. These are the passengers that this actor will have to ensure they arrive at the destination.

To fulfill the “*Diverge passengers to other flights*” sub-goal it is necessary to execute the following three plans:

- ✓ “*Query flight availability*”, obtain a list of flights (even from other airline companies) with available places to the destination of the passengers without flight and their corresponding dates and times.
- ✓ “*Send passengers to hotels*”, if any of the available flights found in the previous plan, are in a different day, the airline company needs to provide hotel accommodations for the passengers affected.
- ✓ “*Check-in passengers*”, the final step will be to check-in the passengers in the available flights found.

Aircraft Maintenance Actor

This actor has the soft-goal “*Provide maintenance services*” and the modeling of this actor is out of the scope of this work. It is represented here due to soft-goal dependencies with the Aircraft Recovery actor and to give a better general overview of the problem.

External Ground Services Actor

This actor has the soft-goal “*Provide ground services*” and the comments made to the previous actor are also applicable to this one.

2.1.2. Analysis

2.1.2.1. Organizations

According to GAIA (23) the first step in performing the analysis of a multi-agent system is to determine whether multiple organizations have to coexist in the system and become autonomous interacting multi-agent systems. Generally speaking and according to the GAIA authors, such sub-organizations can be found when there are portions of the overall system that has any of these conditions:

- a. Exhibit a behavior specifically oriented towards the achievement of a given sub-goal.
- b. Interact loosely with other portions of the system.
- c. Require competences that are not needed in other parts of the system.

From the early-requirements analysis it is possible to determine that there are three candidate sub-organizations that fulfill the first condition and partially, the second one. One thing that is common to all of the sub-organizations is the third condition. Let me point out that, although all three conditions are recognized in each of the identified sub-organizations, it is not a requirement. A sub-organization can be identified even if only one of the conditions exist.

The sub-organizations identified are described in Table 2.

Name	Actor(s)	Description
Crew recovery sub-organization	Crew Recovery	This sub-organization has a clearly sub-goal to achieve, that is, to <i>ensure that every flight has a crew member</i> . It will loosely interact with other portions of system because of the dependencies <i>Flights On Time</i> with the Aircraft Recovery agent.
Aircraft recovery sub-organization	Aircraft Recovery	This sub-organization has a clearly sub-goal to achieve, that is, to <i>ensure all flights on time</i> . It will interact with other portions of system because of the dependencies <i>All Crew On Board</i> with the Crew Recovery agent, <i>Good Maintenance Services</i> with Aircraft Maintenance agent and <i>Good Ground Services</i> with External Ground Services agent.
Passenger recovery sub-organization	Passenger Recovery	This sub-organization has a clearly sub-goal to achieve, that is, to <i>ensure all passengers arrive at the destination</i> . It will interact with other portions of system because of the dependencies <i>Flights On Time</i> with the Aircraft Recovery agent and <i>Good Ground Services</i> with External Ground Services agent.

Table 2 – Sub-organizations in the multi-agent system

2.1.2.2. Environment Model

The first decision to be made in defining the environment model is to distinguish between resources and active components. According to (23), resources might be variables or tuples, “made available to the agents for sensing (e.g., reading their values), for effecting (e.g., changing their values) or for consuming (e.g., extracting them from the environment).” On the other hand, active components are components and services capable of performing complex operations with which agents in the MAS have to interact. The distinction between the two is very important and GAIA gives the following guidelines to do the best decision:

- ✓ “When the role of these active components is simply that of a data provider (consider, for example, a Web server or a DBMS mediating access to a dataset, or a simple computer based sensor), it is better to abstract away from their presence and to model them in terms of resources. The rationale for this is that their presence influences only the mechanisms by which agents retrieve resources (i.e., obtaining the data by requesting a service rather than by performing a sensing operation), not the nature of the resources themselves or the internal activities of the agents. Similar considerations may apply for simple components capable of event-notification services (i.e., upon change of a resource value).”
- ✓ “If the environment contains components and services that are capable of performing complex operations (e.g., active databases, active control systems, humans in-the-loop) then their effects on the agents’ perception of the environment can make it hard to model them as a simple resource repository with identifiable patterns of dynamic change to be sensed by agents (or to interact with based on event-notification mechanisms). In such cases, these components should not be treated as part of the environment but, instead, they should be *agentified*.”

Having these guidelines in mind we have identified the resources and the active components in Table 3.

Type	Name	Description
Active Component	Operational Control Supervisor	Final human authority regarding (38): - Initiate, cancel, consolidate or advance flights. - Wet lease of airplanes. - Exchange of airplane or airplane versions. - Delay flights by more than 15 minutes. - Divert or re-route flights (except in-flight diversion).
Active Component	Operations and Schedule Manager	Final human authority regarding (38): - Monthly plan management. - Daily and weekly operation development. - Crew assignment for unplanned flights and irregularities.
Resource	Crew Sign On Information	Contains information regarding the crew sign on for flights. It will be possible to know if a crew member did not report for duty. It will allow to implement the plan “ <i>query crew no shows</i> ” indicated in the crew recovery goal diagram in Figure 7.
Resource	Pairings Information	Contains information regarding the pairings (and flights) that need to have crew members assigned. It will allow to implement the plan “ <i>query for open flights</i> ” indicated in the crew recovery goal diagram in Figure 7.
Resource	Crew Roster Information	Contains information regarding the roster of all crew members for a specific operational base. It will allow implementing the plans “ <i>query available crew</i> ”, “ <i>query</i>

		<i>stand by crew</i> ” and “ <i>assign crew to pairings</i> ” indicated in the crew recovery goal diagram in Figure 7.
Resource	Airport Flight Information System	Contains information regarding the flights in each airport (Estimated Time of Arrival, Time of arrival, Estimated Time of Departure, etc.). It will allow implementing the plan “ <i>query flight availability</i> ” indicated in the passenger recovery goal diagram in Figure 7.
Resource	Aircraft Roster Information	Contains information regarding the roster of all aircrafts. It will allow implementing the plans “ <i>query aircraft roster</i> ”, “ <i>swap aircrafts</i> ”, “ <i>merger flights</i> ” and “ <i>wet lease airplanes</i> ” indicated in the aircraft recovery goal diagram in Figure 7.
Resource	Reservations Information System	Contains information regarding the passenger reservation status, seating availability, check-in, etc., of all flights in a specific airport. It will allow implementing the plans “ <i>query reservations for check-in status</i> ”, “ <i>query flight availability</i> ”, “ <i>send pax to hotels</i> ” and “ <i>check-in pax</i> ” indicated in the passenger recovery goal diagram in Figure 7.

Table 3 – Identification of resources and active components

From Table 3 we can see that two agents will be created corresponding to the active components *Operational Control Supervisor* and *Operations and Schedule Manager* that will be “agentified”.

For the environment model we need to list or represent the resources and the actions that will be performed to access them, from the environmental perspective. Figure 8 is a diagram that represents the structures of each resource in the environment model and gives an idea of the type of actions required on each resource. It is important to note that the Model Environment, at this stage, is not “normalized” in the usual terms of the database design. That normalization will happen later on, when discussing the implementation issues.

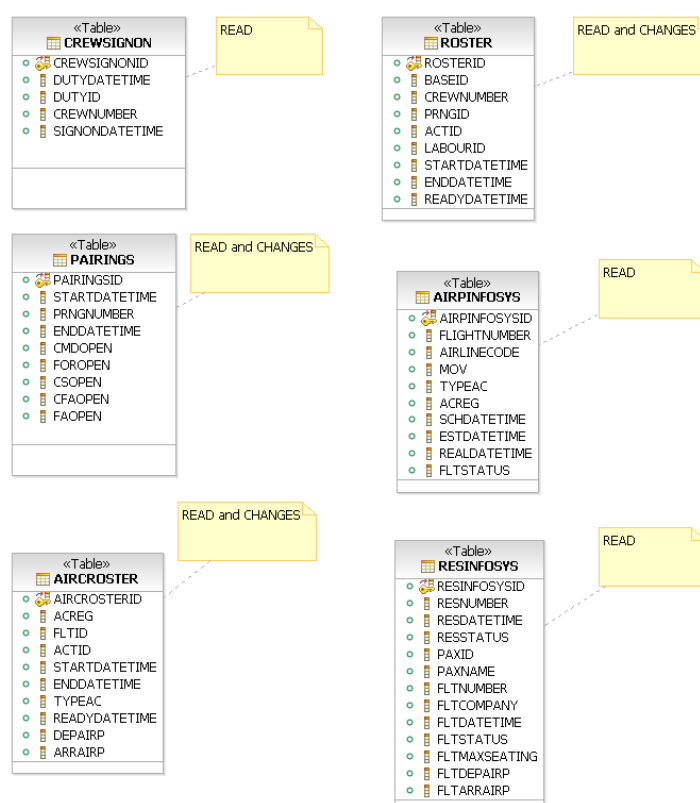


Figure 8 – Diagram of the Environment Model

Table 4 represents, in more detail, how the MAS sub-organization Crew recovery, will access the resources as well as the type of actions that will be performed.

Resource	Attribute(s)	Action and Plan(s)	Description/Condition
CrewSignON	DutyID; CrewNumber;	A: Read P: Query crew no shows	WHERE (<u>Current Date and Time</u>) >= (DutyDateTime) + <u>specific number of min</u> AND (SignOnDateTime) = <u>null</u> .
Pairings	PrgnNumber; CMDOpen; FOOpen; CSOpen; CFAOpen; FAOpen;	A: Read P: Query for open flights	WHERE (<u>Current Date and Time</u>) >= (StartDateTime) – <u>specific number of hours</u> AND (cmdopen + foopen + csopen + cfaopen + faopen) > 0
Pairings	CMDOpen; FOOpen; CSOpen; CFAOpen; FAOpen;	A: Changes P: Assign crew to pairings	Previous value - # <u>crew assigned</u>
Roster	BaseID; CrewNumber;	A: Read P: Query available crew	WHERE ReadyDateTime < <u>Requested Duty Start Date</u> AND StartDateTime > <u>Requested Duty Ready Date</u>
Roster	BaseID; CrewNumber;	A: Read P: Query stand by crew	WHERE ActID = <u>stand by</u> AND StartDateTime <= <u>Requested Duty Start Date</u>
Roster	All record	A: Changes P: Assign crew to pairings	New record inserted and/or deleted existing one to fulfill the plan objective.

Table 4 – Actions performed on the resources and related plans of the Crew Recovery

Table 5 represents how the MAS sub-organization Aircraft recovery, will access the resources.

Resource	Attribute(s)	Actions and Plan(s)	Description/Conditions
AircRoster	AcReg, FltID, TypeAC, FltStatus, DepAirp, ArrAirp.	A: Read P: Query aircraft roster	WHERE (DepAirp= <u>Requested Airport</u> AND RealStrDateTime = <u>null</u> AND EstStrDateTime > StartDateTime) OR (ArrAirp= <u>Requested Airport</u> AND RealEndDateTime = <u>null</u> AND EstEndDateTime > EndDateTime)
AircRoster	All record	A: Read P: Swap aircrafts; Merge flights	WHERE DepAirp = <u>Requested Airport</u> AND TypeAC = <u>Requested Aircraft Family</u> AND StartDateTime >= <u>Requested Flight Date Time</u> AND StartDateTime <= (<u>Requested Flight Date Time</u> – <u>specific number of hours</u>)
AircRoster	All record	A: Changes P: Swap aircrafts; Merge Flights; wet lease planes	New record inserted or deleted existing one to fulfill the plans objectives.

Table 5 – Actions performed on the resources and related plans of the Aircraft Recovery

Finally, Table 6, represents how the MAS the sub-organization Passenger recovery will access the resources.

Resource	Attribute(s)	Action and Plan(s)	Description/Conditions
AirInfoSys	FlightNumber; Mov; TypeAC; ACReg; FltStatus;	A: Read P: Query flight availability	WHERE AirlineCode = <u>Requested Airline Code</u> AND (RealDateTime = <u>null</u> AND EstDateTime > SchDateTime)
ResInfoSys	ResNumber; ResStatus;	A: Read P: Query reservations for	Total Number of Records WHERE Current Date >= (<u>FlightDateTime</u> – <u>specific number of minutes</u>)

	PaxID; PaxName; FltNumber; FltCompany; FltDepAirp; FltArrAirp;	check-in status	AND FltStatus <> <u>departed</u> AND (ResStatus = <u>confirmed</u> OR ResStatus = <u>checked-in</u>) Note: to be used when comparing with the MaxSeating and decide to start recovery action
ResInfoSys	MaxSeating;	A: Read P: Query reservations for check-in status	WHERE FltNumber = <u>specific flight number</u> AND FltCompany = <u>specific flight company</u> AND FltDateTime = <u>specific flight date time</u> Note: to be used when comparing with the TotalNumberRecords and decide to start recovery action
ResInfoSys	FltNumber; FltCompany; FltDateTime;	A: Read P: Query flight availability	WHERE FltStatus = <u>open</u> AND FltDepAirp = <u>request departure airport</u> AND FltArrAirp = <u>request arrival airport</u> AND (MaxSeating – <u>pax checked in</u> > 0)
ResInfoSys	All record	A: Changes P: Check-in pax	New record inserted, updated or deleted related with checking-in passengers in flights after a flight disruption.

Table 6 – Actions performed on the resources and related plans of the Passenger Recovery

2.1.2.3. Preliminary Role Model

At this phase the objective is to identify the basic skills, that is, functionalities and competences, required by the organization to achieve its goals. Those basic skills are the preliminary roles and we need to identify the ones that will be played whatever the organization structure that will be adopted later on during the Architectural Design Phase. GAIA (23) adopts an abstract, semiformal description to express the capabilities and expected behaviors of the preliminary roles. These are represented by two main attribute classes:

1. Permissions.

These attributes have the objective of identify the resources that can be used legitimately to carry out the role and stating the resources limits within which the role must operate. In general, permissions relate agents to the environment in which they are situated. The notation used to represent permissions is the same that we use to represent environmental resources. However, the perspective has changed from an environmental one to what the agents playing the role must be allowed to do to accomplish the role and what they must not be allowed to do.

2. Responsibilities.

These attributes determine the expected behavior of a role. They are divided in two types: *Liveness properties* and *Safety properties* (see pages 344-346 of (23) for more details).

- (i) Liveness properties describe the state of affairs that an agent must bring about, given certain conditions. Liveness expressions are used to define liveness properties (relations) and detail properties related with the dynamics of the organization, that is, how the execution must evolve.
- (ii) Safety properties describe that an acceptable stage of affairs is maintained. Safety expressions are used to define safety properties (constraints) and detail properties that must always be true during the whole life of the MAS.

From the *Actors and Goals* main diagram in Figure 7, we can identify several roles that will exist independently of the final organization of our MAS:

- ✓ RosterCrewMonitor
Role associated with monitoring the crew roster for events related with crew members that do not report for duty and/or flights with open positions.
- ✓ CrewFind
Role associated with finding the best crew member to be assigned to a flight after an event triggered by the RosterCrewMonitor role.
- ✓ CrewAssign
Role associated with assigning the crew member found by the CrewFind role.
- ✓ RosterAircraftMonitor
Role associated with monitoring the departures and arrivals of aircrafts for events related with delay and/or technical problems that might result in actions towards the replacement of the aircraft and/or other actions.

- ✓ AircraftFind
Role associated with finding the best aircraft to be assigned to a flight after an event triggered by the RosterAircraftMonitor role.
- ✓ AircraftAssign
Role associated with assigning the aircraft found by the AircraftFind role.
- ✓ PaxMonitor
Role associated with monitoring the passengers check-in to allow solving problems as a result of flight disruption.
- ✓ PaxFind
Role associated with solving passengers problems after an event has been detected by the PaxMonitor.
- ✓ PaxApply
Role associated with applying the solution found by the PaxFind role.

The *Permissions* (actions allowed on the environment to accomplish the role) have also been identified and are indicated in the Role Schema of each preliminary role, at the end of this section. In the diagram presented in Figure 9 it is possible to have an idea of an environment-preliminary roles diagram:

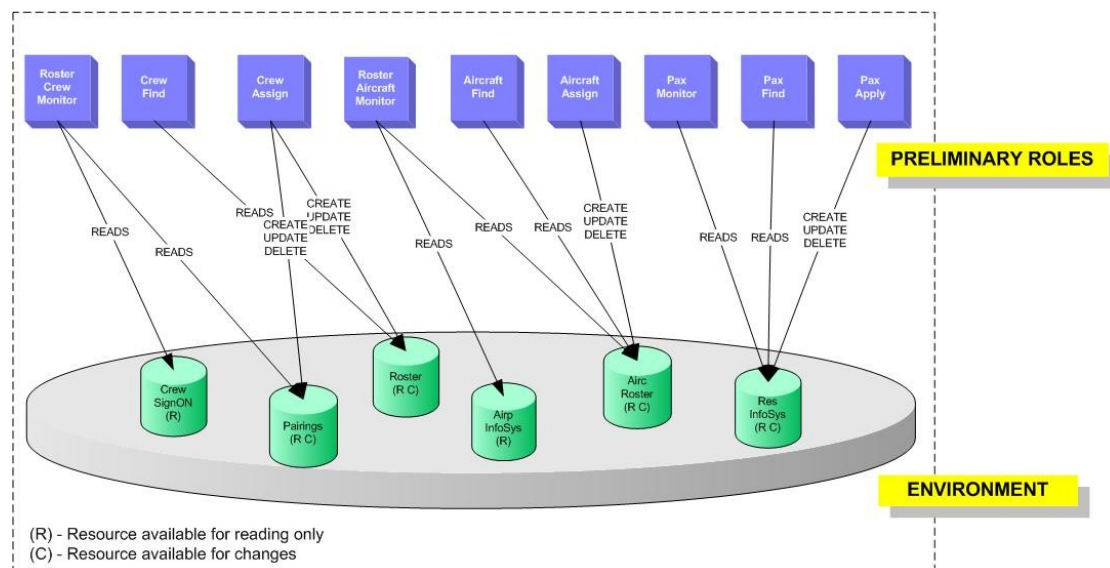


Figure 9 - Environment-Preliminary Roles Diagram

The above diagram helps to identify any inconsistencies between what operations the environment allows (indicated with an R – for reading, and a C – for changes in each resource, information taken from the Environment Model diagram) and what the roles (agents) need or must be allowed to do. As an example, the CrewAssign role needs to create, update and/or delete information from the resources Pairings and Roster. As we can see, the environment allows Changes to be made in those resources. After analyzing this diagram we can see that there are no inconsistencies between the operations allowed by the environment and what the agents need to do. According to GAIA there are other key points that the relations between the environment model and the preliminary role model helps to identify. This can be done using the above Environment-Preliminary Roles diagram in Figure 9:

- i. If a role needs to change a resource that it is only available for reading, it might be necessary to rethought, if possible, the environment model or the capabilities of the roles have to be reduced or re-distributed among roles.
- ii. If we have a physical distributed environment we can determine whether it is reasonable and feasible for a single role to access a wide variety of widely distributed resources, or whether it is more appropriate to divide these capabilities among a set of roles.
- iii. If a role needs to access a variable to which it does not have direct access, a possible solution is to access it via the mediation of a role that does. This may help to identify the need for a given role to be involved in a specific protocol to gain access to that variable. This inter-dependency of the environmental model and of the preliminary interaction model will also be focused in the next section.

Applying the above guidelines to analyze our environment model and our preliminary roles model it is possible to conclude that neither of the situations applies.

The next step is to define the responsibilities of a role, that is, the expected behavior. As stated before, this will be done by defining the two types of properties that compose the responsibilities, namely, liveness properties and safety properties. This will be done using the notation and operators defined in GAIA.

In creating the preliminary roles model it might be impossible to completely define the liveness properties, due to the fact that the organization structure is not yet defined. Regarding the safety properties, it is possible to define some of them in this phase. As an example of the definition of these properties in our case we have the following:

- ✓ Liveness Property:

$$\text{RosterCrewMonitor} = (\text{CheckForNewCrewEvents})^w \parallel (\text{UpdateCrewEventStatus})^w$$

This expression means that the role RosterCrewMonitor performs two activities simultaneously, CheckForNewCrewEvents and UpdateCrewEventStatus (symbol \parallel). Each activity is performed indefinitely (symbol w).
- ✓ Safety Property: *successful_connection_with_Pairings = true*.

This expression means that for the role to perform its task a successful connection to the resource Pairings must exist.

For a better understanding by the reader we adopted the following writing terminology for the analysis and design phase (for the implementation phase it should be used the recommend one according to the development language used):

- ✓ Roles will be written in Pascal Case. Example: RosterCrewMonitor.
- ✓ Protocols will be written in Camel Case. Example: crewRequest.
- ✓ Activities will be written in underlined Pascal Case. Example: SearchCrew.

Finally, we are able to complete the preliminary role model for our case, filling the following Role Schema for each of the roles identified, with the information collected so far. The preliminary role schema for RosterCrewMonitor is presented in Table 7. The rest of the full preliminary role schema can be found in ANNEX B –

PRELIMINARY ROLE MODEL. The preliminary role model will be finished in the Architectural Design, given place to the full Role Model.

Role Schema: RosterCrewMonitor
Description: This preliminary role involves monitoring the crew roster for events related with crew members not reporting for duty and/or flights with open positions. After detecting one of these events it will request to the organization a solution. It should be able to trace previous requests, avoiding duplicates, until receive a message regarding the status of the request.
Protocols and Activities: <u>CheckForNewCrewEvents</u> , <u>UpdateCrewEventStatus</u>
Permissions: reads CrewSignON // to obtain all crew members that do not report for duty reads Pairings // to obtain flights with open positions
Responsibilities Liveness: RosterCrewMonitor = (<u>CheckForNewCrewEvents</u>) ^w (<u>UpdateCrewEventStatus</u>) ^w Safety: - successful_connection_with_CrewSignON = true - successful_connection_with_Pairings = true - new_crew_request <> existing_unclosed_crew_request

Table 7 – RosterCrewMonitor preliminary role

2.1.2.4. Preliminary Interaction Model

The objective of this model is to capture the dependencies and relationships between the various roles in the multi-agent system organization. This is done with one protocol definition for each type of inter role interaction. Because what we have until now is a preliminary role model, the corresponding protocols must also be preliminary. Using GAIA notation for the protocol definition, we define a table, like the one in Table 8, which will be filled with the known attributes until now. The UML 2.0 Sequence Diagrams and UML 2.0 Interaction Overview Diagram can be used to complement the interaction model. We will do that in the Architectural Design phase when we will draw the final Interaction Model. For now, we found useful to complement the *Environment – Preliminary Roles Diagram* of Figure 9 with the preliminary interactions (protocols). The new diagram is presented in Figure 10 below. As you can see it gives a better overview of the all system.

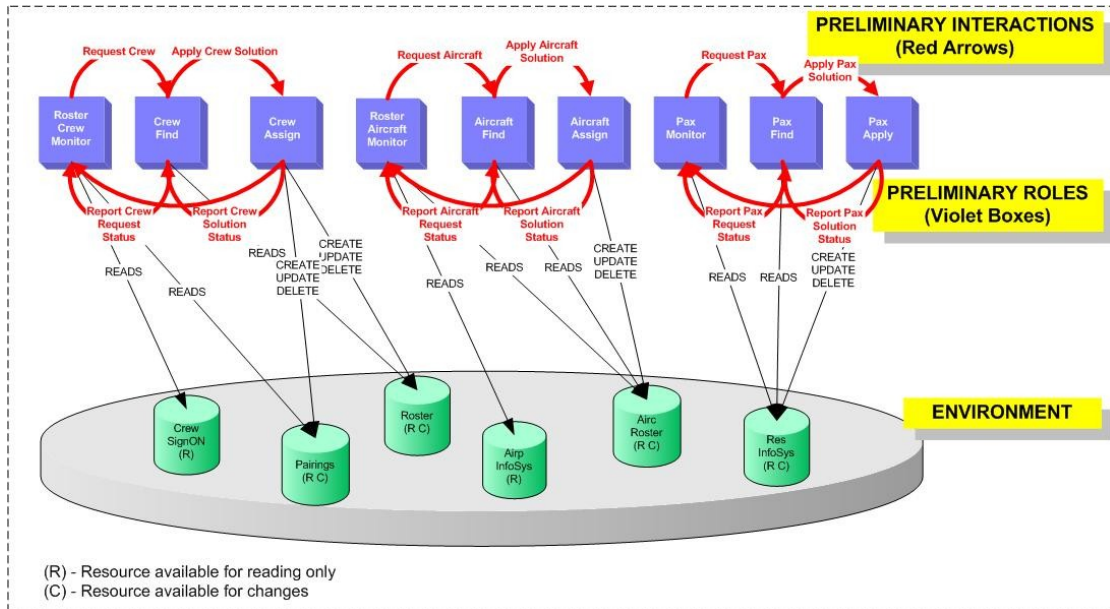


Figure 10 – Environment – Preliminary roles and interaction diagram

The details of the interactions protocols are presented in a table format. Table 8 shows the details of the interaction protocol informCrewEvents. The rest of the preliminary interaction protocol appears in ANNEX C – PRELIMINARY INTERACTION MODEL.

Protocol name: informCrewEvents		
Initiator (role(s)): RosterCrewMonitor	Partner (role(s)): CrewFind	Input: Open position information
Description: After an event has been detected (crew member not reporting for duty and/or flights with open positions) it is necessary to find an available crew member to fill the open position. For that it is necessary to send details about the open position so that an available crew member might be found.		Output: Yes; I will try to find a solution OR No; I cannot process the request (see safety conditions on CrewFind role).

Table 8 – Preliminary protocol definition for informCrewEvents

2.1.2.5. Organizational Rules

According to the authors of GAIA “(...) there may be general relationships between roles, between protocols, and between roles and protocols that are best captured by organizational rules”. Organizational rules are seen as responsibilities of the organization as a whole. In the preliminary roles model we have already defined or approach the roles responsibilities. As in that model, organization rules also have safety and liveness rules (or, as in (39) and as I prefer to see them, constraints and relations respectively):

- ✓ Liveness organizational rules (relations) define “how the dynamics of the organization should evolve over time”. For example, a specific role can be played by an entity only after it has played a given previous role or, in the case of a protocol that it may execute only after another protocol. These liveness expressions can relate to other liveness expressions belonging to different roles.
- ✓ Safety organizational rules (constraints) define “time-independent global invariants for the organization that must be respected”. For example, a specific role can be played by only one entity during the lifetime of the organization or that two roles cannot be played by the same entity. As in the liveness organizational rules the safety rules can relate to safety rules of different roles as well as to expressions of the environment variables in different roles.

The formalism to express these rules can be the same used for the liveness and safety rules for roles (see previous chapter 2.1.2.3. Preliminary Role Model). Liveness and safety organizational rules will be expressed by liveness and safety expressions respectively. As stated previously liveness expressions detail properties related with the dynamics of the organization, that is, how the execution must evolve and safety expressions detail properties that must always be true during the whole life of the MAS.

Using the information we have for our system we can write the organizational rules, as presented in Table 9 and Table 10.

Liveness Rules (relations)	Description
$applyCrewSolution(CrewAssign(crew(x))) \rightarrow reportCrewSolutionStatus(CrewAssign(crew(x)))$	Protocol <i>applyCrewSolution</i> must necessarily be executed by role <i>CrewAssign</i> for a specific crew solution before <i>CrewAssign</i> can execute protocol <i>reportCrewSolutionStatus</i> for that crew solution.
$requestCrew(CrewFind(request(x))) \rightarrow reportCrewRequestStatus(CrewFind(request(x)))$	Protocol <i>requestCrew</i> must necessarily be executed by role <i>CrewFind</i> for a specific request before <i>CrewFind</i> can execute protocol <i>reportCrewRequestStatus</i> for that request.
$applyAircraftSolution(AircraftAssign(aircraft(x))) \rightarrow reportAircraftSolutionStatus(AircraftAssign(aircraft(x)))$	Protocol <i>applyAircraftSolution</i> must necessarily be executed by role <i>AircraftAssign</i> for a specific aircraft solution before <i>AircraftAssign</i> can execute protocol <i>reportAircraftSolutionStatus</i> for that aircraft solution.
$requestAircraft(AircraftFind(request(x))) \rightarrow reportAircraftRequestStatus(AircraftFind(request(x)))$	Protocol <i>informACEvents</i> must necessarily be executed by role <i>AircraftFind</i> for a specific request before <i>AircraftFind</i> can execute protocol <i>reportACEventStatus</i> for that request.
$applyPaxSolution(PaxApply(pax(x))) \rightarrow reportPaxSolutionStatus(PaxApply(pax(x)))$	Protocol <i>applyPaxSolution</i> must necessarily be executed by role <i>PaxApply</i> for a specific pax solution before <i>PaxApply</i> can execute protocol <i>reportPaxSolutionStatus</i> for that pax solution.
$requestPax(PaxFind(request(x))) \rightarrow reportPaxRequestStatus(PaxFind(request(x)))$	Protocol <i>informPaxEvents</i> must necessarily be executed by role <i>PaxFind</i> for a specific request before <i>PaxFind</i> can execute protocol <i>reportPaxEventStatus</i> for that request.

Table 9 – Liveness (Relations) Organizational Rules

Safety Rules (constraints)	Description
$\neg(RosterCrewMonitor \mid CrewFind)$	Role <i>RosterCrewMonitor</i> and role <i>CrewFind</i> can never be played concurrently by the same entity.
$\neg(RosterCrewMonitor \mid CrewAssign)$	Role <i>RosterCrewMonitor</i> and role <i>CrewAssign</i> can never be played concurrently by the same entity.
$\neg(RosterAircraftMonitor \mid AircraftFind)$	Role <i>RosterAircraftMonitor</i> and role <i>AircraftFind</i> can never be played concurrently by the same entity.
$\neg(RosterAircraftMonitor \mid AircraftAssign)$	Role <i>RosterAircraftMonitor</i> and role <i>AircraftAssign</i> can never be played concurrently by the same entity.
$\neg(PaxMonitor \mid PaxFind)$	Role <i>PaxMonitor</i> and role <i>PaxFind</i> can never be played concurrently by the same entity.
$\neg(PaxMonitor \mid PaxAssign)$	Role <i>PaxMonitor</i> and role <i>PaxAssign</i> can never be played concurrently by the same entity.
$\neg(CrewFind \mid AircraftFind)$	Role <i>CrewFind</i> and role <i>AircraftFind</i> can never be played concurrently by the same entity.
$\neg(CrewFind \mid PaxFind)$	Role <i>CrewFind</i> and role <i>PaxFind</i> can never be played concurrently by the same entity.
$\neg(AircraftFind \mid PaxFind)$	Role <i>AircraftFind</i> and role <i>PaxFind</i> can never be played concurrently by the same entity.

Table 10 – Safety (Constraints) Organizational Rules

2.1.3. Architectural Design

Having all the functionality of the MAS as well as the characteristics of the operational environment expressed through the previous documents as a result of the analysis phase, it is time to start the architectural design of the system. Those specifications will be used to identify a way of structuring the MAS organization and to complete the preliminary roles and interaction models. According to (23) “(...) while the analysis phase is mainly aimed at understanding what the MAS will have to be, the design phase is where decisions have to be taken about the actual characteristics of the MAS.” This means that, besides completing and refining the preliminary models, the design will rely in actual decisions about the organizational structure and in modeling the MAS based on the specifications produced.

During this phase, the choice of the organizational structure is very important and will affect the development of the succeeding phases. From the literature (23) it is possible to see that there are several forces that drive the identification of an appropriate organizational structure. The correlation between these forces is best showed through the Figure 11:

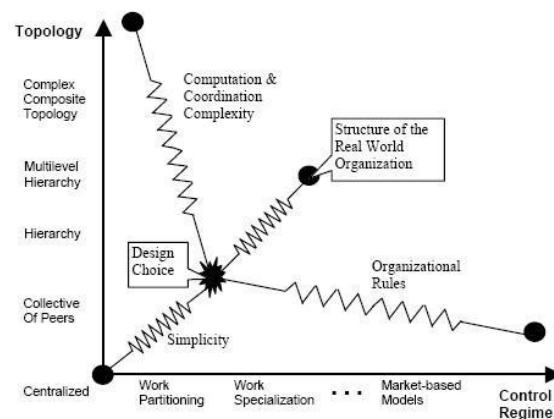


Figure 11 – Forces involved in the identification of the organizational structure

For the organization structure we need to choose the desired topology and the control regime to be applied. This choice is influenced by the computation and coordination complexity of the MAS, the need to respect organizational rules and the necessity of mimic the real world organization. The topology range goes from a more centralized and hierarchical structure to a more distributed and heterarchical one. The control regime range goes from a centralized to a market-based model. As we can see from Figure 11, the choice of a topology derives mainly from the computation and coordination complexity costs that are incurred as the number of members increase (liveness organizational rules may also influence) and the control regime derives mainly from organizational rules (especially safety ones). For a more detailed explanation of the organizational theory please consult the paper (40) from Mark S. Fox.

In the next section we will define and represent the organizational structure for our MAS, based on this correlation of forces as well as in the specification documents that resulted from the analysis phase.

2.1.3.1. Organizational Structure

From the specifications documents of the analysis phase we have the following main requirements to consider for defining the organizational structure, which might have an impact on this decision (for one operational base):

- ✓ The main organization is an operational base with three sub-organizations, as follows:
 - Crew recovery.
 - Aircraft recovery.
 - Passenger recovery.
- ✓ From the environment model we have identified the following active components (resources that will be “agentified”):
 - Operational Control Supervisor (Human authority).
 - Operations and Schedule Manager (Human authority).
- ✓ From the preliminary role model we have identified a requirement that the CrewFind role and AircraftFind role should use different techniques (that is, different algorithms) to find the solutions.
- ✓ From the organizational rules we have identified the following roles that cannot be played concurrently by the same entity:
 - *RosterCrewMonitor* and *CrewFind*;
 - *RosterCrewMonitor* and *CrewAssign*;
 - *RosterAircraftMonitor* and *AircraftFind*;
 - *RosterAircraftMonitor* and *AircraftAssign*;
 - *PaxMonitor* and *PaxFind*;
 - *PaxMonitor* and *PaxAssign*;
 - *CrewFind* and *AircraftFind*;
 - *CrewFind* and *PaxFind*;
 - *AircraftFind* and *PaxFind*.

GAIA does not define a specific notation to represent the organization structure. It suggests a coupled adoption of a formal notation and of a more intuitive graphical representation. The graphical representation can be a simple one (boxes representing roles and annotated arrows representing relations and their types) or one resulting from AUML (41) or from UML 2.0 (35). We have represented the organization structure in a formal notation through a table and in a graphical notation, either a simple one that will include the environment model and another one using UML 2.0.

Taking into consideration the above requirements and changing, as necessary, the previous analysis, we came up with the following organization structure represented in Table 12, Table 13 and Table 14 through a formal notation (one table for each of the sub-organizations identified). Table 11 gives a summary of the topologies and control regimes applied. Please note that the relationships types identified here are neither mutually exclusive (for example, a control relation type may also imply a dependency relation type), nor complete (other types of relations may be identified).

Organization	Topology	Control regime
Base	Multilevel hierarchy	Mixed: cooperative and authoritative
Crew recovery	Multilevel hierarchy	Work specialization and work partitioning (if necessary in the case of the specific problem solver algorithms)
Aircraft Recovery	Multilevel hierarchy	Work specialization and work partitioning (if necessary in the case of the specific problem solver algorithms)
Passenger recovery	Hierarchy	Work specialization

Table 11 – Summary of topologies and control regimes used

Statement/Comment
$\forall i, \text{OperationsScheduleManager} \xrightarrow{\text{control}} \text{CrewAssign}[i]$
Means that the role <i>OperationsScheduleManager</i> has an authoritative relationship with role <i>CrewAssign</i> , controlling, in this case, all the actions of role <i>CrewAssign</i> . Specifically, role <i>CrewAssign</i> needs approval from <i>OperationsScheduleManager</i> before applying the solution.
$\forall i, \text{OperationsScheduleManager} \xrightarrow{\text{depends_on}} \text{CrewFind}[i]$
Means that role <i>OperationsScheduleManager</i> relies on resources or knowledge (a solutions found to solve a crew recovery problem) from role <i>CrewFind</i> to accomplish is task (that is, to authorize or not authorize the assignment of a specific solution).
$\forall i, j, \text{CrewFind}[i] \xrightarrow{\text{depends_on}} \text{RosterCrewMonitor}[j]$
Means that role <i>CrewFind</i> relies on resources or knowledge (an event related with a crew problem) from role <i>RosterCrewMonitor</i> to accomplish is task (that is, to find a solution to the crew problem).
$\forall i, j, \text{CrewFind}[i] \xrightarrow{\text{control}} \text{TapCRHeuristic}[j]$
Means that the role <i>CrewFind</i> controls the actions of role <i>TapCRHeuristic</i> . Specifically, role <i>TapCRHeuristic</i> receives the indication to find the solution using its expertise and, later, role <i>CrewFind</i> will decide from all the solutions received from all the others roles (<i>OtherCRAAlgorithm</i> , for example) which one will forward to the <i>OperationsScheduleManager</i> . This relation and the next one implement the Work Specialization control regime.
$\forall i, j, \text{CrewFind}[i] \xrightarrow{\text{control}} \text{OtherCRAAlgorithm}[j]$
This one is similar to the previous statement.
$\forall i, j, \text{TapCRHeuristic}[i] \xrightarrow{\text{controls}} \text{TapCRPart}[j]$
It is not clear at this time if this relation and correspondent topology and control regime will be implemented. It is dependent on a more in depth analysis of each of the algorithms that will be used. If necessary, the Workload Partition control regime in a hierarchy topology will be used to obtain a sensible load balance of work between all the members.
$\text{OperationsScheduleManager} \xrightarrow{\text{peer}} \text{OperationalControlSupervisor}$
At an operational base level role <i>OperationsScheduleManager</i> and <i>OperationalControlSupervisor</i> are peers in which they collaborate to solve problems.

Table 12 - Organization structure for Crew Recovery sub-organization (formal notation)

Statement/Comment
$\forall i, \text{OperationalControlSupervisor} \xrightarrow{\text{control}} \text{AircraftAssign}[i]$
Means that the role <i>OperationalControlSupervisor</i> has an authoritative relationship with role <i>AircraftAssign</i> , controlling, in this case, all the actions of role <i>AircraftAssign</i> . Specifically, role <i>AircraftAssign</i> needs approval from <i>OperationalControlSupervisor</i> before applying the solution. Please note that role <i>OperationalControlSupervisor</i> is shared between this sub-organization and Passenger Recovery sub-organization.
$\forall i, \text{OperationalControlSupervisor} \xrightarrow{\text{depends_on}} \text{AircraftFind}[i]$
Means that role <i>OperationalControlSupervisor</i> relies on resources or knowledge (a solutions found to solve an aircraft recovery problem) from role <i>AircraftFind</i> to accomplish is task (that is, to authorize or not authorize the assignment of a specific solution).
$\forall i, j, \text{AircraftFind}[i] \xrightarrow{\text{depends_on}} \text{RosterAircraftMonitor}[j]$
Means that role <i>AircraftFind</i> relies on resources or knowledge (an event related with an aircraft problem) from role <i>RosterAircraftMonitor</i> to accomplish is task (that is, to find a solution to the crew problem).

$\forall i, j, AircraftFind[i] \xrightarrow{control} TapARHeuristic[j]$
Means that the role <i>AircraftFind</i> controls the actions of role <i>TapARHeuristic</i> . Specifically, role <i>TapARHeuristic</i> receives the indication to find the solution using its expertise and, later, role <i>AircraftFind</i> will decide from all the solutions received from all the others roles (<i>OtherARAlgorithm</i> , for example) which one will forward to the <i>OperationalControlSupervisor</i> . This relation and the next one implement the Work Specialization control regime.
$\forall i, j, AircraftFind[i] \xrightarrow{control} OtherARAlgorithm[j]$
This one is similar to the previous statement.
$\forall i, j, TapARHeuristic[i] \xrightarrow{controls} TapARPart[j]$
It is not clear at this time if this relation and correspondent topology and control regime will be implemented. It is dependent on a more in depth analysis of each of the algorithms that will be used. If necessary, the Workload Partition control regime in a hierarchy topology will be used to obtain a sensible load balance of work between all the members.
$OperationalControlSupervisor \xrightarrow{peer} OperationsScheduleManager$
At an operational base level role <i>OperationalControlSupervisor</i> and <i>OperationsScheduleManager</i> are peers in which they collaborate to solve problems.

Table 13 - Organization structure for Aircraft Recovery sub-organization (formal notation)

Statement/Comment
$\forall i, OperationalControlSupervisor \xrightarrow{control} PaxApply[i]$
Means that the role <i>OperationalControlSupervisor</i> has an authoritative relationship with role <i>PaxApply</i> , controlling, in this case, all the actions of role <i>PaxApply</i> . Specifically, role <i>PaxApply</i> needs approval from <i>OperationalControlSupervisor</i> before applying the solution. Please note that role <i>OperationalControlSupervisor</i> is shared between this sub-organization and Aircraft Recovery sub-organization.
$\forall i, OperationalControlSupervisor \xrightarrow{depends_on} PaxFind[i]$
Means that role <i>OperationalControlSupervisor</i> relies on resources or knowledge (a solutions found to solve a pax recovery problem) from role <i>PaxFind</i> to accomplish is task (that is, to authorize or not authorize the assignment of a specific solution).
$\forall i, j, PaxFind[i] \xrightarrow{depends_on} PaxMonitor[j]$
Means that role <i>PaxFind</i> relies on resources or knowledge (an event related with a pax problem) from role <i>PaxMonitor</i> to accomplish is task (that is, to find a solution to the pax problem).

Table 14 - Organization structure for Passenger Recovery sub-organization (formal notation)

To help visualize the organization structure it is possible to draw a simple diagram of the above structure. To have a full picture we have coupled the organization structure with the environment model from previous diagrams. The result is presented in Figure 12.

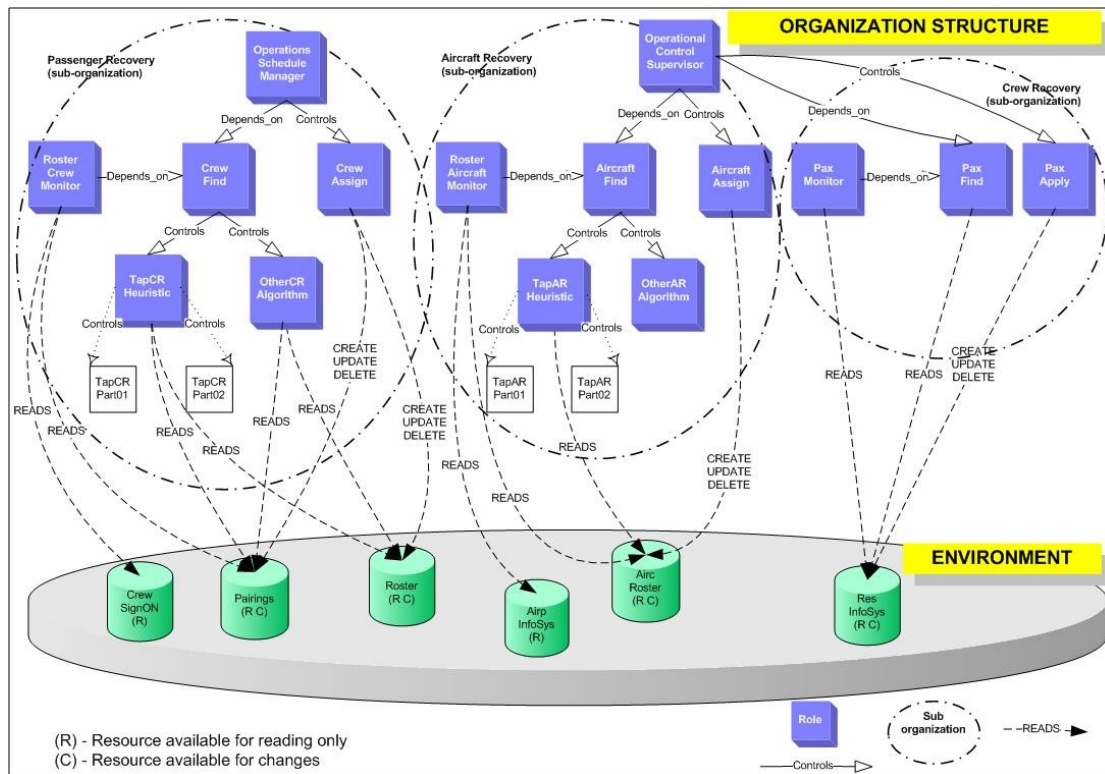


Figure 12 - Organization Structure with Environment Model


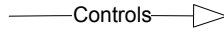


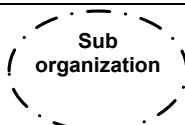

	Represents a role.
	This annotated arrow connects the roles and represents the relation and their types.
	This annotated arrow represents a possible relation and their type. Means that decision is still pending of further analysis.
	This annotated arrow represents the influence of the roles in the environment including the type of action in that environment.
	This ellipsis represents the sub-organization detected during analysis.
	Represents a resource in the environment including the type of action allowed. R = read, C = change.

Table 15 – Legend of Organization Structure diagram

Although GAIA does not define a specific notation for the graphical representation of the organization structure we have decided to try to do it using UML 2.0 as in Figure 13 and Figure 14.

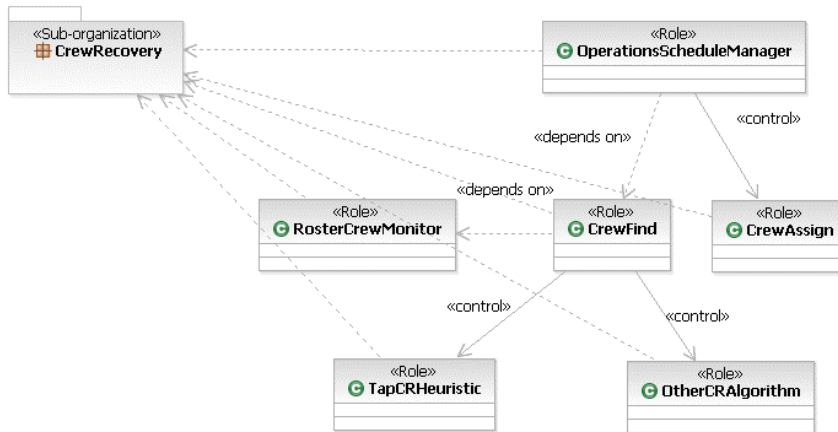


Figure 13 - Organization Structure for Crew Recovery in UML 2.0

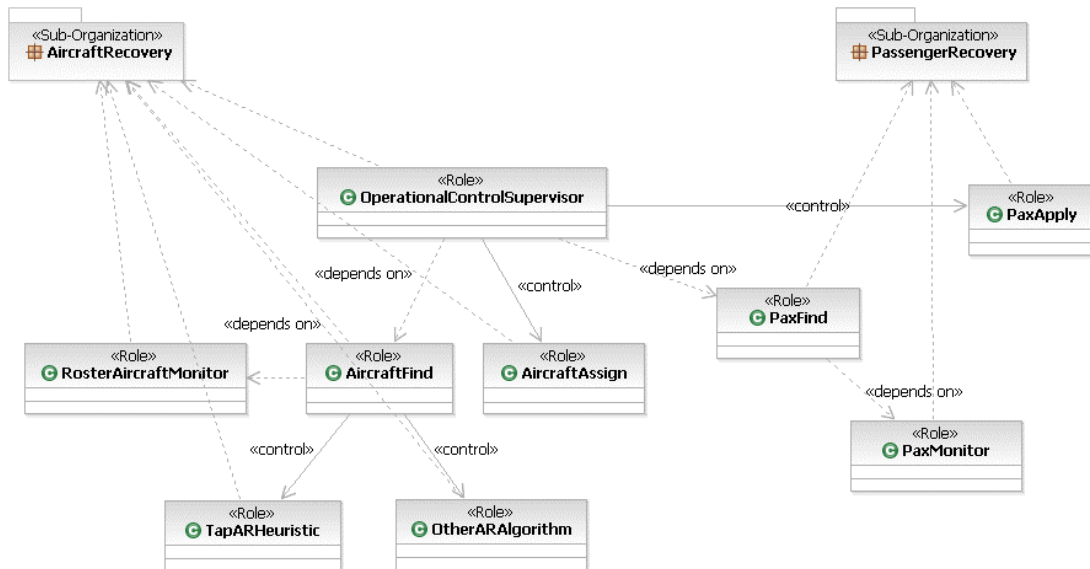


Figure 14 - Organization Structure for Aircraft and Passenger Recovery in UML 2.0

To be able to represent the organization structure in UML 2.0 we have made some mappings between the abstractions used here and the UML 2.0 artifacts as well as created some stereotypes.

Organization Abstraction

We have mapped this abstraction to a *package*. In UML packages provide a way to group related elements. Using package diagrams it is possible to visualize dependencies between parts of the system. If we see an organization as a package we can take advantage of these characteristics and model them using package diagrams. For the goal of the organization we can use a note or constraint to represent it. We have created a sub-organization stereotype associated to the package metaclass of UML.

Depends on Abstraction

We have mapped this abstraction to a *dependency* relationship. The dependency relationship in UML is the weakest it is possible to define. A dependency between

classes means that one class uses, or has knowledge of, another class. They are typically read as "...uses a...". A depends on relation between two roles usually means that one role relies on resources or knowledge from the other role. We have created a depends on stereotype associated to the dependency metaclass of UML.

Controls Abstraction

We have mapped this abstraction to an *association* relationship. *Associations* relationships in UML are stronger than dependencies and typically indicate that one class retains a relationship to another class over an extended period of time. They are typically read as "...has a...". A *control* relation between two roles usually means that one role has an authoritative relationship with the other role, controlling their actions. We have created a *control* stereotype associated to the association metaclass of UML.

Peer Abstraction

We have also mapped this abstraction to a *dependency* relationship. A *peer* relation between two roles usually means that they are at the same level and collaborate to solve problems. We have created a *peer* stereotype associated to the dependency metaclass of UML.

Role Abstraction

We have mapped this abstraction to a *class*. A class represents a group of things that have a common state and behavior. A class can represent a tangible and concrete concept, such as an invoice, or it may be abstract, such as a document. Roles are functionalities and competences, that we need to characterize. At this point we found this representation useful. However, as stated in (35) "roles cannot be modeled in the necessary details with any UML 2.0 diagram". We have created a *role* stereotype associated to the class metaclass of UML.

2.1.3.2. Role Model and Interaction Model

Now that the Organization Structure is defined it is possible to complete the role model and the interaction model. Some roles interaction result from the organization topology defined and the protocols that need to be executed result from the control regime defined. So, the tasks that are necessary to be performed to complete both models are:

- ✓ Complete all *activities* in which a role will be involved, including its liveness and safety responsibilities.
- ✓ Define *organizational roles*, that is, those whose presence was not identified during analysis and that result directly from the adopted organization structure.
- ✓ Complete the definition of protocols specifying which roles the protocol will involve.
- ✓ Define *organizational protocols*, that is, those whose identification derives from the adopted organization structure.

Starting with the preliminary role model we should now fully identify all activities and services that are necessary, including new roles that directly derive from the adoption of a specific organization structure.

Regarding the interaction model we should complete the preliminary interaction model, completing the identification of the involved roles, including new protocols resulting from the adoption of the control regime.

It is important to clearly preserve the distinction between those characteristics that are intrinsic, that is, independent of the use of the role and/or protocol in a specific organization structure (usually the ones identified in the preliminary model) from the extrinsic, that is, the ones that derive from the adoption of a specific organizational structure (usually the new ones identified during the definition of the organization structure). The identification of the intrinsic and extrinsic characteristics will have an important role in the perspective of reuse and design for change.

Once completed the role and interaction model and according to GAIA “they will represent an operational description of the MAS organization that could be effectively exploited, possibly with the support of the environment model, for the detailed design of the MAS.”

We have to say that it was in this phase that the most important modeling decisions were made. It was here that we have changed more the previous preliminary decisions, because we started to have a closer view into the final model of the system. Because of that, improvements in the design were achieved.

Table 16 shows the role schema for RosterCrewMonitor and Table 17 shows the protocol definition for informsCrewEvents, both in a table format. The rest of the role model and interaction model are presented in ANNEX D – ROLE MODEL and ANNEX E – INTERACTION MODEL, respectively.

Role Schema: RosterCrewMonitor (RCM)
Description: Monitors the crew roster for events related with crew members not reporting for duty and/or flights with open positions. After detecting one of these events it will request to the organization a solution. Traces previous requests and avoids duplicates, until receive a message regarding the status of the request.
Protocols and Activities: <u>CheckForNewCrewEvents</u> , <u>UpdateCrewEventStatus</u> , <u>informsCrewEvent</u> , <u>reportCrewEventStatus</u>
Permissions: reads CrewSignON // to obtain all crew members that do not report for duty reads Pairings // to obtain flights with open positions create, read, update Crew Events Queue Data Class // keeps a record of events status
Responsibilities Liveness: RosterCrewMonitor = (<u>CheckForNewCrewEvents</u> ^W . <u>informsCrewEvent</u>) ^W (<u>reportCrewEventStatus</u> ^W . <u>UpdateCrewEventStatus</u>) ^W Safety: - successful_connection_with_CrewSignON = true - successful_connection_with_Pairings = true - successful_connection_with_CrewEvents = true - new_crew_request <> existing_unclosed_crew_request

Table 16 - RosterCrewMonitor (RCM) role

Protocol name: informCrewEvents		
Initiator (role(s)): RosterCrewMonitor (RCM)	Partner (role(s)): CrewFind (CF)	Input: Open position information
Description: After an event has been detected (crew member not reporting for duty and/or flights with open positions) it is necessary to find an available crew member to fill the open position. For that it is necessary to send details about the open position so that an available crew member might be found.		Output: Yes; I will try to find a solution OR No; I cannot process the request (see safety conditions on CrewFind role).

Table 17 – Protocol definition for informsCrewEvents

A UML 2.0 representation of the interaction protocol informCrewEvents is presented in Figure 15 .

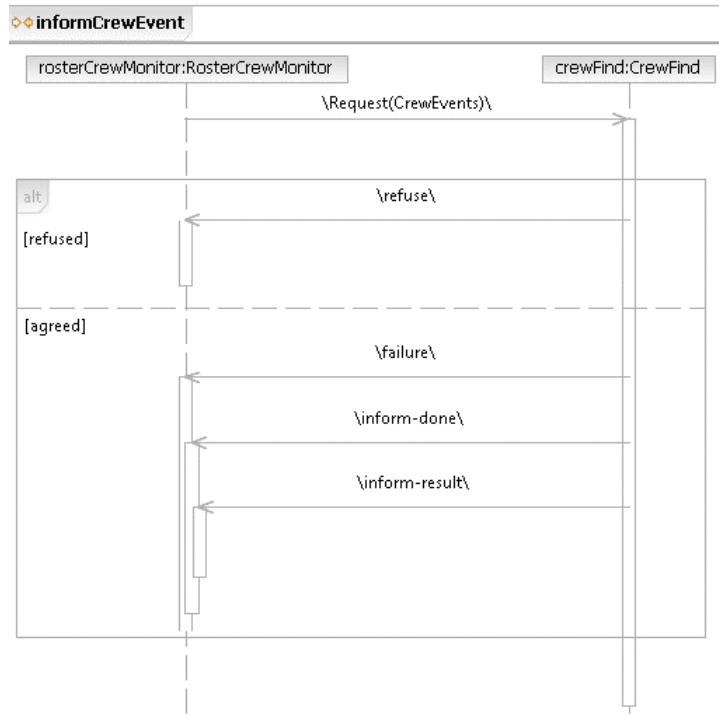


Figure 15 - UML 2.0 Interaction Diagram for informCrewEvents protocol

With the information from the role model and interaction model it is possible to draw a simple graphical representation of the Environment, Role and Interaction models. It is represented in Figure 16.

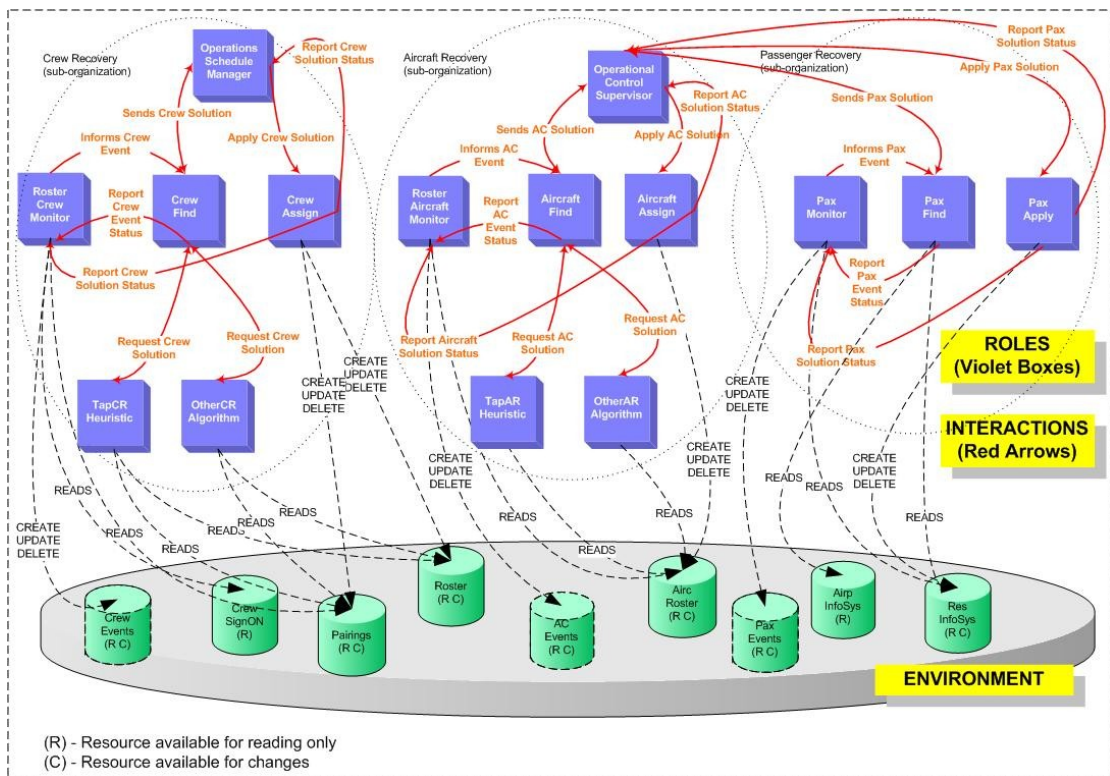


Figure 16 - Environment - Role and Interactions diagram

A possible representation of this diagram in UML 2.0 is the one presented in Figure 17 (Crew recovery Sub-organization) and Figure 18 (Aircraft and Pax Sub-organization).

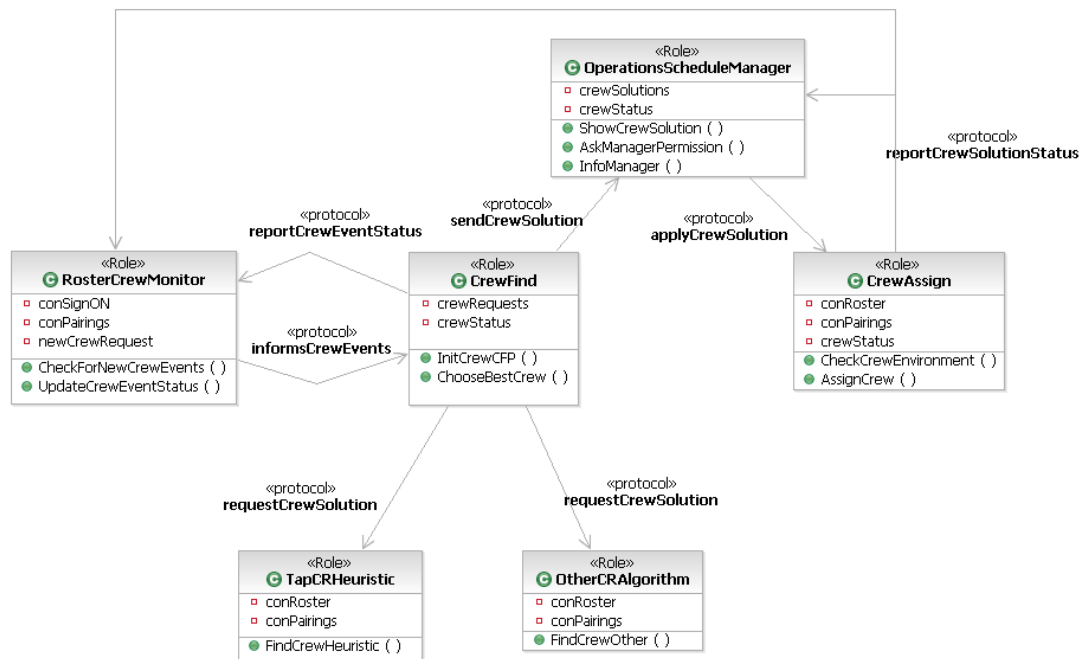


Figure 17 - UML 2.0 representation for Role and Interaction Model of Crew recovery

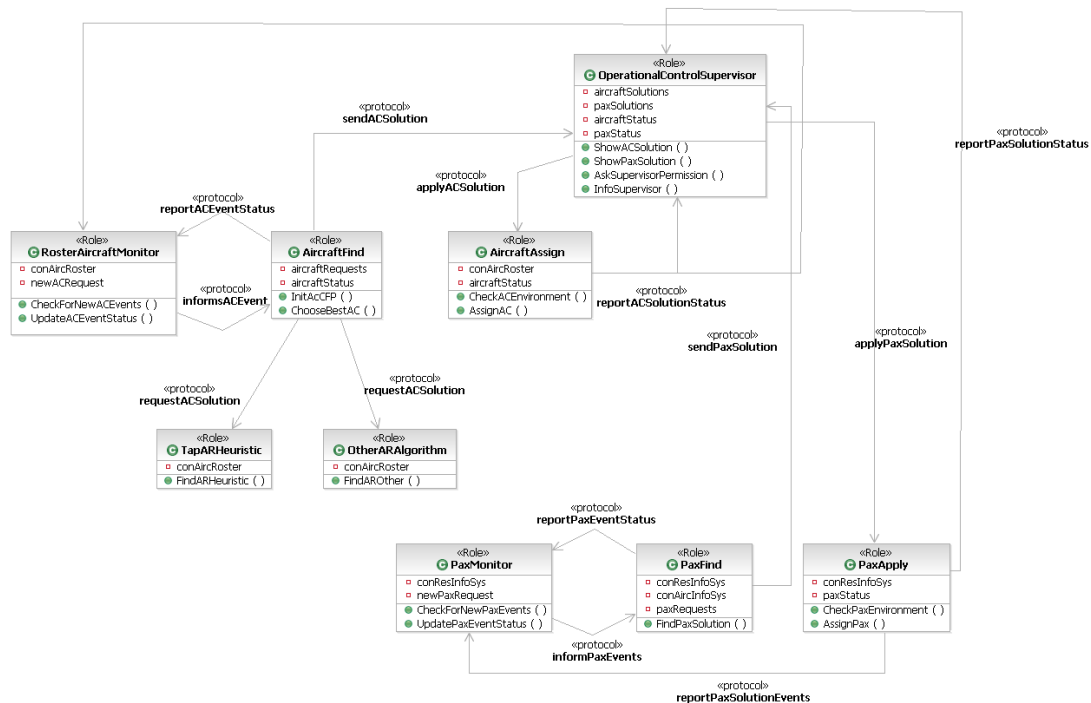


Figure 18 - UML 2.0 representation for Role and Interaction Model of Aircraft and Pax recovery

Besides the mappings we have done for the organization structure representation in UML 2.0, in this diagram we have done the following:

Role Abstraction

We complement the previous considering the usual attributes of the class as being part of the safety properties. For example, the role *RosterCrewMonitor* has three attributes: *conSignOn*, *conPairings* and *newCrewRequest*. Those attributes are part of the following safety expressions, respectively: *successful_connection_with_CrewSignOn = true*, *successful_connection_with_Pairings = true* and *new_crew_request <> existing_unclosed_crew_request*. The activities are indicated as methods. In this example, the role has two activities: *CheckForNewCrewEvents* and *UpdateCrewEventStatus*.

Protocol Abstraction

The activities that involve interactions with other roles (protocols) are represented by an *Association* relationship in UML. We have created a *protocol* stereotype associated to the association metaclass in UML.

Although for the implementation phase, some of these mappings might not be the appropriate ones, it did help us to visualize the organization with their roles, activities and protocols, using a commercially available tool.

2.1.4. Detailed Design

2.1.4.1. Agent Model

Regarding the Agent Model, we can make a one-to-one correspondence between roles and agent classes. However, there are some advantages in trying to find a better mapping. The best one is to try to compact the design by reducing the number of classes and instances leading to a reduction in conceptual complexity. This has to be done without:

- ✓ Affect the organizational efficiency,
- ✓ Violating organizational rules and
- ✓ Creating “bounded rationality” problems (that is, cannot exceed the amount of information it is able to store and process in a given amount of time).

GAIA does not specify any special notation for showing the Agent model. It can be a simple table, specifying for each class which roles will map to it and, additionally, indicate the instances of each class that will appear in the MAS. First we draw a table (Table 18) with the information and, then, at the end of this chapter we will present a possible representation using UML 2.0 diagrams, following the suggestion presented in (35).

Agent Class/Roles	Description
$OpMonitor^{1..n} \xrightarrow{play} RosterCrewMonitor,$ $RosterAircraftMonitor,$ $PaxMonitor$	Means that agent class OpMonitor will be defined to play the roles RosterCrewMonitor, RosterAircraftMonitor and PaxMonitor, and that we will have between one to n instance of this class in our MAS (n depends on the number of operational bases defined).
$OpAssign^{1..n} \xrightarrow{play} CrewAssign,$ $AircraftAssign,$ $PaxApply$	Means that agent class OpAssign will be defined to play the roles CrewAssign, AircraftAssign and PaxApply, and that we will have between one to n instance of this class in our MAS (n depends on the number of operational bases defined).
$OpManager^1 \xrightarrow{play} \rightarrow$ $OperationsScheduleManager,$ $OperationalControlSupervisor$	Means that agent class OpManager will be defined to play the roles OperationsScheduleManager and OperationalControlSupervisor, and we will have one instance of this class in our MAS.
$OpCRFind^{1..n} \xrightarrow{play} CrewFind$	Agent class OpCRFind will play role CrewFind and the number of instances will depend of the number of bases.
$OpARFind^{1..n} \xrightarrow{play} AircraftFind$	Similar to previous.
$OpPXFind^{1..n} \xrightarrow{play} PaxFind$	Similar to previous.
$OpTapCRH^{1..n} \xrightarrow{play} TapCRHeuristic$	Similar to previous.
$OpTapARH^{1..n} \xrightarrow{play} TapARHeuristic$	Similar to previous.
$OpOtherCRA^{1..n} \xrightarrow{play} OtherCRAAlgorithm$	Similar to previous.
$OpOtherARA^{1..n} \xrightarrow{play} OtherARAAlgorithm$	Similar to previous.

Table 18 - Agent Model

2.1.4.2. Service Model

For the Service Model we have to identify the services that each agent will have. These services are derived from the protocols, activities and liveness properties of the roles that the agent implements. As a rule of thumb, there will be one service for each parallel activity of execution that the agent has to execute. However, it might be possible to introduce more services even for sequential activities of execution, especially in the cases where it is necessary to represent different phases of the agent execution.

The service model requires that, for each service that may be performed by an agent, four properties are identified: inputs, outputs, pre-conditions and post-conditions.

The inputs and outputs are derived from the interaction model (protocols) and from the environment model. If the service involves the elaboration of data and the exchange of knowledge between agents, the inputs and outputs will come from the protocols. If the service involves the evaluation and modification of the environment resources, the inputs and outputs will come from the environment.

The pre and post conditions represent restrictions on the execution and completion, respectively, of the services. They derive from the role safety properties as well as from the organizational rules. They can involve restrictions on the availability and on the specific values assumed by the environment resources and/or by the data and knowledge of other agents.

Applying the above guidelines to our specific problem we have obtained the Service Model. The service model for the agent class OpMonitor is represented in Table 19. The rest of the model is represented in ANNEX F – SERVICE MODEL.

Service	Input	Output	Pre-condition	Post-condition
Monitor crew events	Current date, crew slack time, pairing slack time.	A list of: DutyID, crew number, prng number, list of open positions, eventID.	Successful connection with CrewSignON and Pairings resources	A new crew event has to be different from an existing unclosed event.
Update crew event status	EventID, event status	Number of records updated.	Successful connection with CrewEvents resource	A successful update of the CrewEvents resource
Monitor aircraft events	Current date, departure airport, arrival airport	A list of: AcReg, FltID, TypeAC, FltStatus	Successful connection with AircRoster.	A new aircraft event has to be different from an existing unclosed event.
Update aircraft event status	EventID, event status	Number of records updated.	Successful connection with ACEvents resource	A successful update of the ACEvents resource
Monitor pax events	Current date, pax slack time	A list of: ResNumber ResStatus, PaxID, PaxName, FltNumber, FltCompany, FltDepAirp, FltArrAirp.	Successful connection with ResInfoSys.	A new pax event has to be different from an existing unclosed event.
Update pax event status	EventID, event status	Number of records updated.	Successful connection with PaxEvents resource	A successful update of the PaxEvents resource.

Table 19 – Service Model for agent class OpMonitor

2.1.4.3. UML 2.0 Representation

A possible representation of the agent model and the service model in a UML 2.0 diagram is presented in Figure 19 and Figure 20, respectively.

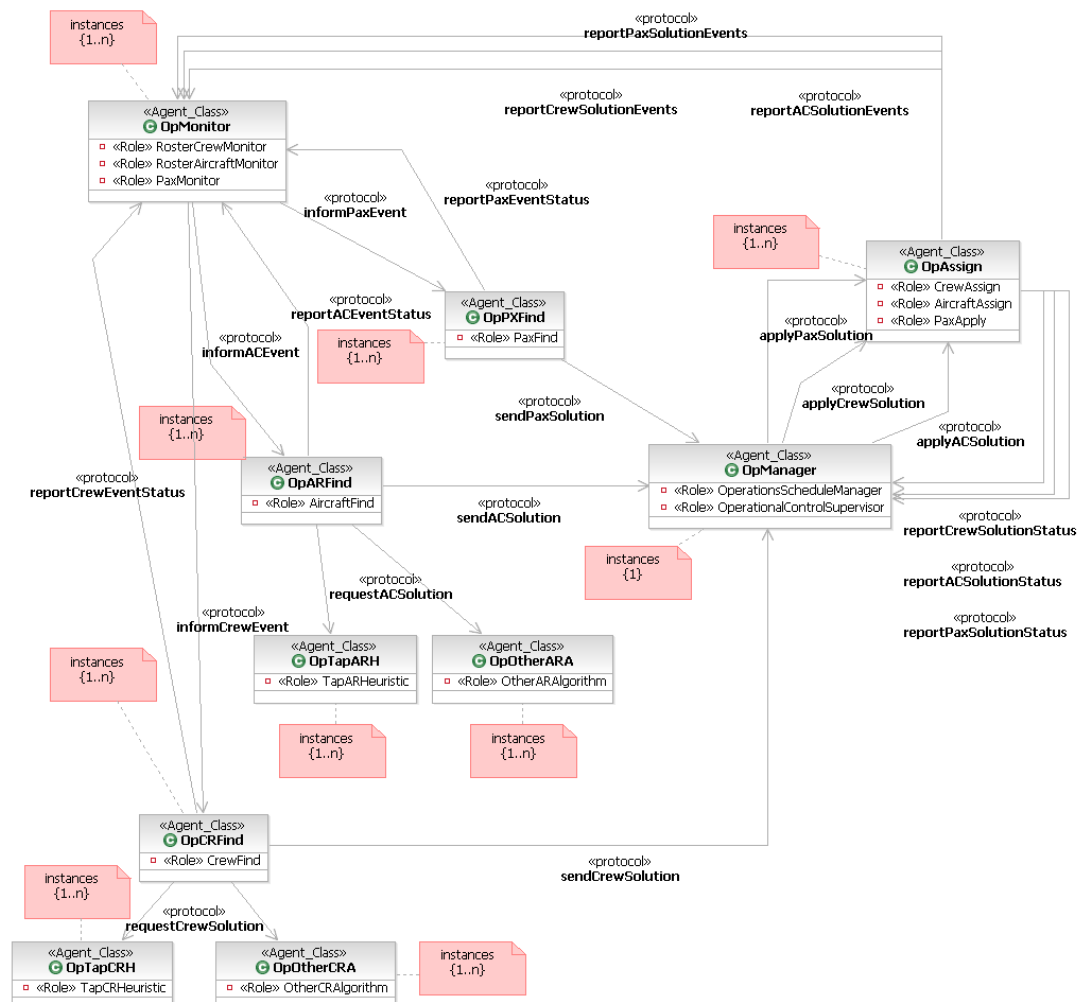


Figure 19 - UML 2.0 Representation of the Agent Model

Agent Class Abstraction

We have mapped this abstraction to a *class* and created an *Agent Class* stereotype associated to the class metaclass in UML. To identify the roles that each agent class implements we have created a *role* stereotype associated to the property metaclass in UML. The instances of the agent class are represented using *Constraints*.

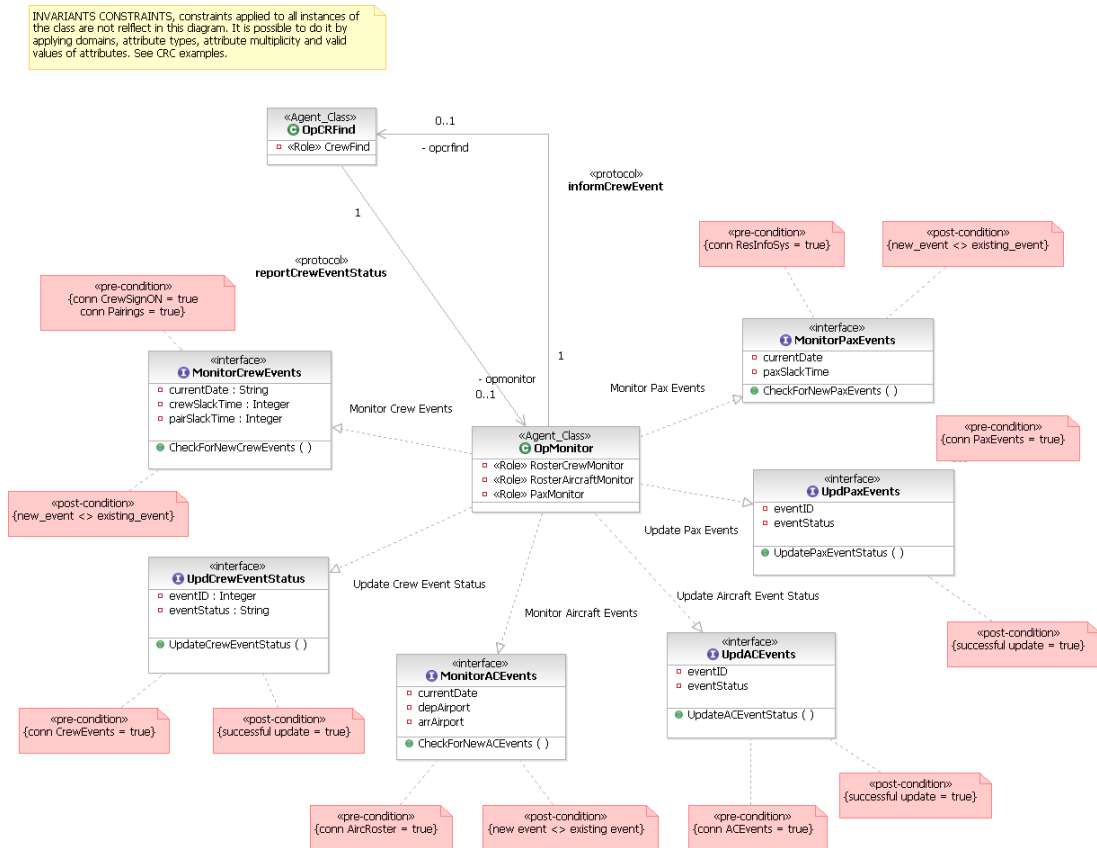


Figure 20 - Service Model (Partial) for Agent Class OpMonitor

Services Abstraction

We mapped the services abstraction to an interface. In UML an interface is a classifier that has declarations of properties and methods but no implementations. It provides a contract that a classifier that provides an implementation of the interface must obey. The inputs are represented as properties of the interface and the method represents the service that needs to be implemented. The outputs are what the method returns. For example, the interface MonitorCrewEvents represents the service with the same name and attributes currentDate, crewSlackTime and pairSlackTime are the inputs. CheckForNewCrewEvents is the operations to be implemented. The agent class OpMonitor realizes that interface by providing an implementation for the operations and properties (the dashed line starting at the agent class to the interface, with a closed arrowhead at the end, shows this realization).

Pre/Post-conditions

We present the pre/post-condition using constraints, associated with the specific interface. Example for MonitorCrewEvents interface:

```

«pre-condition»
{conn CrewSignON = true; conn Pairings = true}
«post-condition»
{new_event <> existing_event}
  
```

It is important to note that Invariant Constraints, constraints applied to all instances of the class are not reflected in this diagram. It is possible to do it by applying domains, attribute types, and attribute multiplicity and valid values of attributes.

2.2. Implementation

Due to the use of JAVA and JADE as the implementation language and middleware, it is necessary to map between the detailed design obtained from GAIA and language/middleware that we use. Before starting the implementation it is necessary to perform the following steps:

1. Model the interaction between the several agents (in terms of communications and how to represent the content of messages), identifying the proper concepts and actions and defining them as classes. Decide which of the two methods (serialized java objects or extensions of predefined JADE classes) will be the ideal one to use. The idea is to start from the interaction model, environment model and agent and service models and use them in this task.
2. Define a notation to be used for the names of Agents, Services and Protocols according to the implementation language and their best practices.
3. Create a table, relating each one of the services in the service model (use also the relevant and appropriate liveness expressions on the role schema) with the possible JADE behavior to use, according to the necessary activities to be performed. This table can also include the JADE interaction protocol to be used (if that is the case).
4. Create a table, defining for each one of the interactions protocol in the model, the necessary ACL performatives and why. This table should reflect the choice of using an existing JADE protocol (see task 3) or the choice of building one.

2.2.1. Concepts and Actions

For the agents to communicate in a way that makes sense for them, they must share the same language, vocabulary and protocols. Although JADE already supports a certain degree of commonality, by following the FIPA standards, it will be necessary to define the vocabulary and semantics for the content of the messages that will be exchanged by the agents in our system. JADE provides three ways to implement communication between agents:

1. Use of strings to represent the content of messages. It is convenient when the content of the messages is atomic data, but not in the case of abstract concepts, objects or structured data. In such cases, the strings need to be parsed to access its various parts.
2. Exploit of Java technology to transmit serialized Java objects directly as the content of messages. It is a convenient method for a local application where all agents are implemented in Java. One inconvenience is that messages are not readable by humans.
3. Definition of the objects to be transferred as extension of predefined classes so that Jade can encode and decode messages in a standard FIPA format. This allows Jade agents to interoperate with other agent systems.

The first thing to do, before deciding which of the methods to use, is to identify the pertinent concepts and actions and to define these as classes. After reviewing the interaction, environment, agent and services models, created during the analysis and

design of our system, we identified the necessary concepts and actions represented in Table 20.

Name	Type	Description
CrewEvent	Concept	Characterizes a crew event that initiates the process of crew recovery.
AircraftEvent	Concept	Characterizes an aircraft event that initiates the process of aircraft recovery.
PassengerEvent	Concept	Characterizes a passenger event that initiates the process of passenger recovery.
UpdateEventStatus	Action	Action of making the status update of a Crew/Aircraft or Passenger event due to a crew/aircraft/passenger recovery process.
CrewSolutionList	Concept	Characterizes a list of crew solutions proposed by the agents that are specialists in crew recovery and that corresponds to the CFP initiated after a crew event has been detected. The agent CrewFindAgent will choose the best solution from this list.
AircraftSolutionList	Concept	Characterizes a list of aircraft solutions proposed by the agents that are specialists in aircraft recovery and that corresponds to the CFP initiated after an aircraft event has been detected. The agent AircFindAgent will choose the best solution from this list.
CrewSolution	Concept	Characterizes the crew solution choose by agent CrewFindAgent and that will be presented to the ManagerAgent for authorization.
AircraftSolution	Concept	Characterizes the aircraft solution choose by agent AircFindAgent and that will be presented to the ManagerAgent for authorization.
PassengerSolution	Concept	Characterizes the passenger solution found by agent PaxFindAgent and that will be presented to the ManagerAgent for authorization.
ApplyCrewSolution	Action	Action of applying the crew solution after it has been authorized by the ManagerAgent.
ApplyAircraftSolution	Action	Action of applying the aircraft solution after it has been authorized by the ManagerAgent.
ApplyPassengerSolution	Action	Action of applying the passenger solution after it has been authorized by the ManagerAgent.

Table 20 – Concepts and actions to be represented as classes

Due to the fact that our agents are developed in Java and that, in this version, our MAS is not an open system and it does not need to interoperate with other agents systems, we will choose to pass the content of the messages with objects. In the future and if the interoperability with other agents will be necessary, it is easy to change the objects and extend the class Ontology predefined in Jade.

2.2.2. Agents, Protocols and Services Notations

The names used in the implementation and corresponding mapping to the names used in analysis and design are in Table 21.

Type	Design Name	Implementation Name
Agents	OpMonitor	MonitorAgent
	OpCRFind	CrewFindAgent
	OpTapCRH	CrewHeuristicAgent
	OpOtherCRA	CrewOtherAlgorAgent
	OpARFind	AircFindAgent
	OpTapARH	AircHeuristicAgent
	OpOtherARA	AircOtherAlgorAgent
	OpPXFind	PaxFindAgent
	OpManager	ManagerAgent
	OpAssign	AssignAgent
Protocols	informCrewEvent	inform-crew-event
	reportCrewEventStatus	request-crew-event-status-update
	informACEvent	inform-aircraft-event
	reportACEventStatus	request-aircraft-event-status-update
	informPaxEvent	inform-pax-event
	reportPaxEventStatus	request-pax-event-status-update
	requestCrewSolution	crew-solution-negotiation
	sendCrewSolution	query-crew-solution-authorization
	requestACSolution	aircraft-solution-negotiation
	sendACSolution	query-aircraft-solution-authorization
	sendPaxSolution	query-pax-solution-authorization
	applyCrewSolution	request-apply-crew-solution
	reportCrewSolutionStatus	request-crew-solution-status-update
	applyACSolution	request-apply-aircraft-solution
	reportACSolutionStatus	request-aircraft-solution-status-update
	applyPaxSolution	request-apply-pax-solution
	reportPaxSolutionStatus	request-pax-solution-status-update
Services	Monitor crew events	MonitorCrewEvents
	Update crew event status	UpdateCrewEventStatus
	Monitor aircraft events	MonitorAircEvents
	Update aircraft event status	UpdateAircEventStatus
	Monitor pax events	MonitorPaxEvents
	Update pax event status	UpdatePaxEventStatus
	Check crew environment	CheckCrewEnvironment
	Assign crew	AssignCrew
	Check aircraft environment	CheckAircEnvironment
	Assign aircraft	AssignAircraft
	Check pax environment	CheckPaxEnvironment
	Assign pax	AssignPax
	Obtain crew solution authorization	ObtainCrewAuthorization
	Request crew solution application	RequestApplyCrew
	Obtain AC solution authorization	ObtainAircAuthorization
	Request AC solution application	RequestApplyAircraft
	Obtain pax solution authorization	ObtainPaxAuthorization
	Request pax solution authorization	RequestApplyPax
	Find crew solution	FindCrew
	Choose best crew solution	ChooseBestCrew
	Find aircraft solution	FindAircraft
	Choose best aircraft solution	ChooseBestAircraft
	Find pax solution	FindPax
	Obtain crew list from heuristic	ObtainHeuristicCrewList
	Obtain crew list from other algorithm	ObtainAlgorCrewList
	Obtain aircraft list from heuristic	ObtainHeuristicAircList
	Obtain aircraft list from other algorithm	ObtainAlgorAircList

Table 21 – Names to be used in implementation

2.2.3. Services and JADE behaviors

All services are implemented with JADE behaviors that will “run” inside a JADE CyclicBehaviour or that will extend the CyclicBehaviour. This is necessary because all agents will be running indefinitely, as it is possible to infer from the liveness expressions of the roles that each agent represents. The agents perform indefinitely some services (for example, monitoring) and/or waiting for a message to act (for example, messages that initiate interactions protocols that they need to be part of). Table 22 shows the mapping between JADE behaviors and the services implemented, including protocols ID’s.

Service	JADE Behaviour	FIPA/JADE IP	Protocol(s) ID
MonitorCrewEvents	Ticker	fipa-request	inform-crew-event
UpdateCrewEventStatus	OneShot	fipa-request	request-crew-event-status-update
MonitorAircEvents	Ticker	fipa-request	inform-aircraft-event
UpdateAircEventStatus	OneShot	fipa-request	request-aircraft-event-status-update
MonitorPaxEvents	Ticker	fipa-request	inform-pax-event
UpdatePaxEventStatus	OneShot	fipa-request	request-pax-event-status-update
CheckCrewEnvironment	Simple	fipa-request fipa-request	request-apply-crew-solution [request-crew-solution-status-update]
AssignCrew	Simple	fipa-request	request-crew-solution-status-update
CheckAircEnvironment	Simple	fipa-request fipa-request	request-apply-aircraft-solution [request-aircraft-solution-status-update]
AssignAircraft	Simple	fipa-request	request-aircraft-solution-status-update
CheckPaxEnvironment	Simple	fipa-request fipa-request	request-apply-pax-solution [request-pax-solution-status-update]
AssignPax	Simple	fipa-request	request-pax-solution-status-update
ObtainCrewAuthorization	Simple	fipa-query	query-crew-solution-authorization
RequestApplyCrew	Simple	fipa-request fipa-request	request-apply-crew-solution request-crew-solution-status-update
ObtainAircAuthorization	Simple	fipa-query	query-aircraft-solution-authorization
RequestApplyAircraft	Simple	fipa-request fipa-request	request-apply-aircraft-solution request-aircraft-solution-status-update
ObtainPaxAuthorization	Simple	fipa-query	query-pax-solution-authorization
RequestApplyPax	Simple	fipa-request fipa-request	request-apply-pax-solution request-pax-solution-status-update
FindCrew	Simple	fipa-request fipa-contract-net	inform-crew-event crew-solution-negotiation
ChooseBestCrew	Simple	fipa-query fipa-request	query-crew-solution-authorization request-crew-event-status-update
FindAircraft	Simple	fipa-request fipa-contract-net	inform-aircraft-event aircraft-solution-negotiation
ChooseBestAircraft	Simple	fipa-query fipa-request	query-aircraft-solution-authorization request-aircraft-event-status-update
FindPax	Simple	fipa-request fipa-query fipa-request	inform-pax-event [query-pax-solution-authorization] request-pax-event-status-update
ObtainHeuristicCrewList	Sequential	fipa-contract-net	crew-solution-negotiation
ObtainAlgorCrewList	Sequential	fipa-contract-net	crew-solution-negotiation
ObtainHeuristicAircList	Sequential	fipa-contract-net	aircraft-solution-negotiation
ObtainAlgorAircList	Sequential	fipa-contract-net	aircraft-solution-negotiation

Table 22 – Mapping of JADE behaviors and Protocols to be used to implement services

2.2.4. ACL Performatives

Table 23 indicates for each interaction protocol the performatives to be used.

Interaction Protocol	Performatives to be used	Comments
inform-crew-event	request; refuse; agree; failure; inform (done); inform (result);	Performatives used in FIPA-Request Protocol.
request-crew-event-status-update	request; refuse; agree; failure; inform (done); inform (result);	Performatives used in FIPA-Request Protocol.
inform-aircraft-event	request; refuse; agree; failure; inform (done); inform (result);	Performatives used in FIPA-Request Protocol.
request-aircraft-event-status-update	request; refuse; agree; failure; inform (done); inform (result);	Performatives used in FIPA-Request Protocol.
inform-pax-event	request; refuse; agree; failure; inform (done); inform (result);	Performatives used in FIPA-Request Protocol.
request-pax-event-status-update	request; refuse; agree; failure; inform (done); inform (result);	Performatives used in FIPA-Request Protocol.
crew-solution-negotiation	cfp; refuse; propose; reject_proposal; accept_proposal; failure; inform (done); inform (result);	Performatives used in FIPA ContracNet Protocol
query-crew-solution-authorization	query_if; refuse; agree; failure; inform(t/f); inform (result);	Performatives used in FIPA Query Protocol
aircraft-solution-negotiation	cfp; refuse; propose; reject_proposal; accept_proposal; failure; inform (done); inform (result);	Performatives used in FIPA ContracNet Protocol
query-aircraft-solution-authorization	query_if; refuse; agree; failure; inform(t/f); inform (result);	Performatives used in FIPA Query Protocol
query-pax-solution-authorization	query_if; refuse; agree; failure; inform(t/f); inform (result);	Performatives used in FIPA Query Protocol
request-apply-crew-solution	request; refuse; agree; failure; inform (done); inform (result);	Performatives used in FIPA-Request Protocol.
request-crew-solution-status-update	request; refuse; agree; failure; inform (done); inform (result);	Performatives used in FIPA-Request Protocol.
request-apply-aircraft-solution	request; refuse; agree; failure; inform (done); inform (result);	Performatives used in FIPA-Request Protocol.
request-aircraft-solution-status-update	request; refuse; agree; failure; inform (done); inform (result);	Performatives used in FIPA-Request Protocol.
request-apply-pax-solution	request; refuse; agree; failure; inform (done); inform (result);	Performatives used in FIPA-Request Protocol.
request-pax-solution-status-update	request; refuse; agree; failure; inform (done); inform (result);	Performatives used in FIPA-Request Protocol.

Table 23 – Performatives to be used for each interaction protocol

2.2.5. Crew Recovery Sub-Organization Implementation

As stated before, the MAS has three sub-organizations: Crew, Aircraft and Pax Recovery. These sub-organizations have their own architecture with their specialized agents, that is, the agents that implement the algorithms specialized in solving problems. Figure 21 shows the architecture for *Crew Recovery* in a UML diagram, taken from the Agent Model presented in chapter 2.1.4.1. Agent Model. The architecture for *Aircraft Recovery* and *Pax Recovery* are very similar.

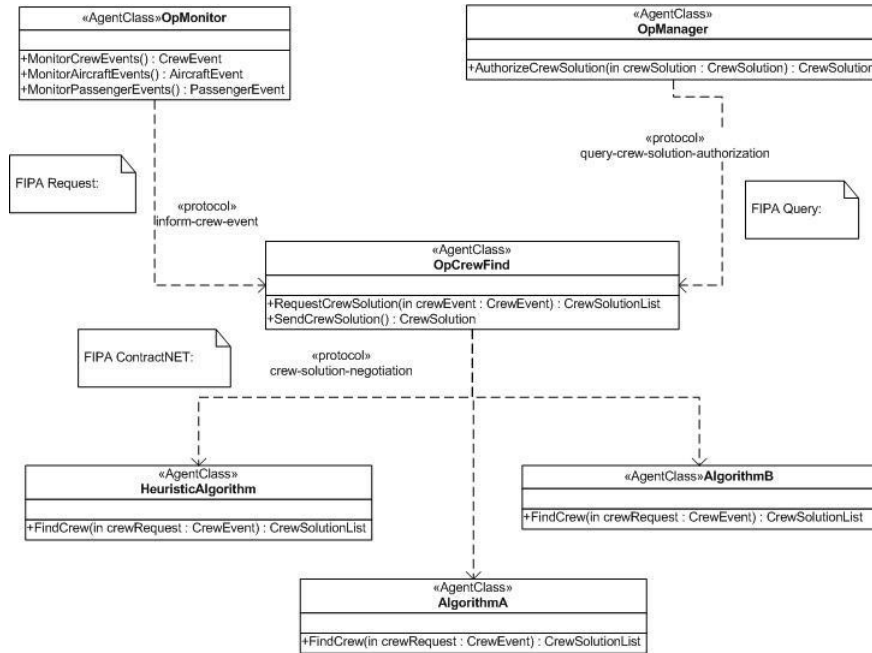


Figure 21 - Crew Recovery Architecture

The agent class *OpMonitor* is responsible for monitoring any crew events, for example, crew members that did not report for duty or duties with open positions, that is, without any crew member assigned to a specific role on board (e.g., captain or flight attendant). When an event is detected, the service *MonitorCrewEvents* will initiate the protocol *inform-crew-event* (FIPA Request) informing the *OpCrewFind* agent. The message will include the information necessary to characterize the event. This information is passed as a serializable object of the type *CrewEvent*. Figure 22 shows the attributes of the *CrewEvent* class.



Figure 22 Crew Event

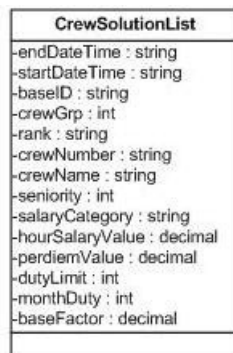


Figure 23 Crew Solution List

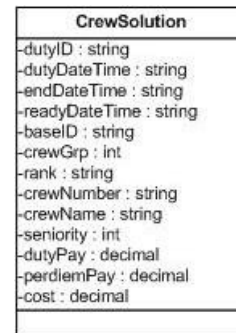


Figure 24 Crew Solution

The *OpCrewFind* agent detects the message and will start a CFP (call for proposal) through the *crew-solution-negotiation* protocol (FIPA contractNET) requesting to the

specialized agents *HeuristicAlgorithm*, *AlgorithmA* and *AlgorithmB* of any operational base of the airline company, a list of solutions for the problem. Each agent implements a different algorithm specific for this type of problem. When a solution is found a serializable object of the type *CrewSolutionList* is returned in the message as an answer to the CFP. Figure 23 shows the attributes of the *CrewSolutionList* class.

The *OpCrewFind* agent collects all the proposals received and chooses the best one according to the algorithm in Table 24.

```

foreach item in CrewSolution list
  totalDuty = monthDuty+credMins
  if (totalDuty-dutyLimit) > 0
    credDuty = totalDuty-dutyLimit
  else
    credDuty = 0
  end if
  perdiemDays = (endDateTime-dutyDateTime)
  perdiemPay = perdiemDays*perdiemValue
  dutyPay = credDuty*(hourSalaryValue/60)
  cost = (dutyPay+perdiemPay)*baseFactor
end foreach
order all items by cost desc
select first item on the list

```

Table 24 - Crew Best Proposal Choice Algorithm

The algorithm in Table 24 is implemented in the service *SendCrewSolution* and produces a list ordered by the cost (a multi-criteria cost) that each solution represents. Table 25 explains each of the computed values in the algorithm in Table 24.

<i>totalDuty</i>	Monthly duty minutes of the proposed crew member after assigning the new duty
<i>credDuty</i>	Number of minutes to be paid case the crew member exceeds the monthly duty limit
<i>dutyPay</i>	Cost of duty computed according to the hour salary of the crew member
<i>perdiemDays</i>	Number of days of work for the specific duty
<i>perdiemPay</i>	Cost of duty computed according to the perdiem value of the crew member
<i>baseFactor</i>	If the crew member belongs to the same operational base where the problem happened, the value is equal to one. Otherwise, it will have a value greater than one.
Cost	The sum of the cost of the perdiem plus duty multiplied by the base factor.

Table 25 - Crew Best Proposal Algorithm computed values

The first solution of the list in descendant order by cost corresponds to the less expensive one. The *SendCrewSolution* service initiates the protocol *query-crew-solution-authorization* (FIPA Query) querying the *OpManager* agent for authorization. The message includes the serializable object of the type *CrewSolution* as shown in Figure 24.

2.2.6. JADE Implementation Examples

The implementation code of the CrewEvent concept is presented in Figure 25.

```

/**
 * CONCEPT: Charcaterizes a crew event that
 * initiates the process of crew recovery
 */
import java.io.Serializable;
public class CrewEvent implements Serializable{

    private String dutyID; //Identification of the DUTY (an activity other than a pairing)
    private String crewNumber; // Crew number of the crew member that did not report for duty
    private String prngNumber; // identification of the Pairing (the one with open positions)
    private String openPositions; // Code that corresponds to the open positions (CPT; FO;)
    private String eventID; // Code that corresponds to the eventID generated automatically by the MAS

    /*
     * getters and setters for the attributes
     */
    public String getDutyID() {
        return dutyID;
    }
    public void setDutyID(String thedutyID) {
        dutyID = thedutyID;
    }
    public String getCrewNumber() {
        return crewNumber;
    }
    public void setCrewNumber(String thecrewNumber) {
        crewNumber = thecrewNumber;
    }
    public String getPrngNumber() {
        return prngNumber;
    }
    public void setPrngNumber(String theprngNumber) {
        prngNumber = theprngNumber;
    }
    public String getOpenPositions() {
        return openPositions;
    }
    public void setOpenPositions(String theopenPositions) {
        openPositions = theopenPositions;
    }
    public String getEventID() {
        return eventID;
    }
    public void setEventID(String theeventID) {
        eventID = theeventID;
    }
}

```

Figure 25 - CrewEvent concept implementation code

Figure 26 shows the implementation of the OpMonitor Agent Class its roles in JADE.

```
package ops.crew.recovery;

import jade.core.*;
/**
 * Agent Class OpMonitor
 * implements roles RosterCrewMonitor
 * and services MonitorCrewEvents, UpdateCrewEventStatus
 * using JADE behaviours Ticker and OneShot.
 */
public class OpMonitor extends Agent {
    // declare agent level data classes

    protected void setup() {
        // activate MonitorCrewEvents service
        addBehaviour(new MonitorCrewEventsBehaviour(this, 1000));
        // activate UpdateCrewEventStatus service
        addBehaviour(new UpdateCrewEventStatusBehaviour());
    }
}
```

Figure 26 - OpMonitor Agent Class implementation

Figure 27 shows how the service MonitorCrewEvents is implemented through a JADE TickerBehaviour.

```

package ops.crew.recovery;

import java.util.Date;
import jade.core.Agent;
import jade.core.behaviours.*;

/**
 * MonitorCrewEvents service implemented through
 * JADE Behaviour TickerBehaviour.
 * It is executed every "period" milliseconds
 */
public class MonitorCrewEventsBehaviour extends TickerBehaviour {
    public MonitorCrewEventsBehaviour(Agent a, long period) {
        super(a, period);
    }
    // Monitors the CrewSignON and Pairings resource
    protected void onTick() {
        // Inputs of the Service
        Date dt = new Date("2006-07-01"); // date to monitor
        int crwSlack = 10; // 10 minutes slack time for crew members
        int pairSlack = 10; // 10 minutes slack time for pairings

        // Starts the service with the inputs
        CrewEvent ce = monitorResources(dt, crwSlack, pairSlack);
        // If the CrewEvent Concept, that is, the output has information
        // it is necessary to start the interaction protocol and
        // request a crew solution
        if (ce != null) {
            //
            // Starts Interaction Protocol INFORM-PAX-EVENT
            // because it was detected an event, that is,
            // a crew member that did not report for duty
            // and/or a pairing with open positions
        }
    }
    // method to implement the SQL code to get information
    // from the environment
    private CrewEvent monitorResources(Date dt, int crwSlack, int pairSlack) {
        CrewEvent ce = new CrewEvent();
        //
        // Perform SQL code to monitor CrewSignON and Pairings resource
        // returning the CrewEvent filled with information
        //
        return ce;
    }
}

```

Figure 27 - MonitorCrewEvents service implementation

3. Results

In chapter 1.5. Hypothesis and Predictions we stated several hypothesis. We could not make all the necessary experiments to prove all hypotheses. However, regarding hypothesis two, we were able to make experiments regarding crew recovery problems. The hypothesis to be evaluated was the following:

“The objective of solving the crew recovery problems as fast as possible with the less cost as possible, will be much easier to achieve if we take advantage of the fact that the crew members belong to different operational bases and if we consider payroll information and other costs in the decision process. We predict that if we solve the problems first with local resources and then with non-local resources, taking into account payroll information like *hour salary* and *perdiem value* of each crew rank, and costs related with hotels and *extra-crew travel*, the solution will be faster to find and less expensive”.

To evaluate our hypothesis we have setup a scenario that includes 3 operational bases (A, B and C). Each base includes their crew members each one with a specific roster. The data used corresponded to the real operation of June 2006 of base A. We have simulated a situation where 15 crew members, with different ranks, did not report for duty in base A. A description of the information collected for each event is presented in Table 26.

Attribute	Description
Event ID	A number that represents the ID of the event. For tracking purposes only
Duty Date Time	The start date and time of the duty in UTC for which the crew did not report.
Duty ID	A string that represents the ID of the duty for which the crew did not report.
End Date Time	The end date and time of the duty in UTC for which the crew did not report.
Ready Date Time	The date and time at which the crew member is ready for another duty after this one.
Delay	The delay of the crew member. We have considered 10 minutes in our scenario.
Credit Minutes (Cred Mins)	The minutes of this duty that will count for payroll.
Crew Group (Crew Grp)	The crew group (Technical = 1; Cabin = 2) that the crew member belongs to.
Crew Rank (Rank)	The crew rank that the crew member belongs to. CPT = Captain; OPT = First Officer; CCB = Chief Purser; CAB = Purser.
Crew Number	The employee number.
Crew Name	The employee name.
Base ID	The base where the event happened. We considered all events in base A.
Open Positions	The number of missing crews for this duty and rank. We used a fixed number of 1.

Table 26 - Description of the information collected for each event

The events did not happen at the same day and each one corresponds to a crew member that did not report for a specific duty in a specific day. Table 27 show the data for each of the events created. As you can see we have omitted the information regarding *Delay*, *Base ID* and *Open Positions* because we have used fixed values as indicated in Table 26.

	duty DateTime	dutyID	end DateTime	ready DateTime	cred Mins	crew Grp	rank	crew Number	crew Name
1	05-06 07:25	1ORY149S	05-06 13:35	06-06 01:35	370	2	CAB	80	John A
2	05-06 07:25	1ORY149S	05-06 13:35	06-06 01:35	370	2	CAB	45	Mary A
3	05-06 07:25	1ORY85P	05-06 13:35	06-06 01:35	370	1	CPT	35	Anthony
4	15-06 04:10	2LIS24X	16-06 16:15	17-06 04:15	1757	2	CAB	99	Paul M
5	15-06 04:10	3LIS25X	15-06 09:20	15-06 21:20	632	2	CAB	56	John B
6	15-06 12:50	2LHR63P	16-06 20:45	17-06 08:45	1549	1	CPT	57	Paul S
7	15-06 12:50	2LHR63P	16-06 20:45	17-06 08:45	1549	1	OPT	53	Mary S
8	15-06 14:15	1LHR31P	15-06 20:55	16-06 08:55	843	2	CCB	23	Sophie
9	15-06 15:25	2LHR19P	16-06 20:45	17-06 08:45	1341	2	CCB	34	Angel
10	15-06 15:25	1ZRH12X	17-06 09:30	17-06 21:30	1318	1	CPT	32	Peter B
11	25-06 05:20	1LIS16S	25-06 15:05	26-06 03:05	585	2	CAB	20	Paul G
12	25-06 05:20	1LIS16S	25-06 15:05	26-06 03:05	585	2	CAB	10	Alice
13	25-06 05:20	1LIS158T	25-06 15:05	26-06 03:05	585	2	CAB	15	Daniel
14	25-06 06:15	3LIS174S	27-06 16:15	28-06 04:15	1258	2	CAB	71	George
15	25-06 14:20	4LIS50A	28-06 19:40	29-06 07:40	219	1	OPT	65	Allan

Table 27 - Crew Event data used in this scenario

For example, the event 10 corresponds to the following situation:

Crew Peter B, with number 32 and rank CPT (captain) belonging to the crew group 1 (technical crew), did not report for the duty with ID 1ZRH12X with briefing time (duty date time) at 15:25 in 15-06-2006. The event was created after a 10 minutes delay and happened at base A. It is necessary to find another crew member to be assigned to this duty. The duty ends at 09:30 on 17-06-2006 and the crew member assigned to this duty will be ready for another one at 21:30 in 17-06-2006. The duty will contribute with 1318 minutes (21h58) for the payroll. The new crew member must belong to the same rank and group.

For a better understanding of the data, Table 28 shows the *Crew Event Data Distribution* according to the Crew Group and Crew Rank.

	Elements
Total Records	15
BY CREW GROUP	
Technical Crew	5
Cabin crew	10
BY CREW RANK	
CPT (Captain)	3
OPT (First Officer)	2
CCB (Chief Purser)	2
CAB (Purser)	8
CAB	8

Table 28 - Crew Event Data distribution

Methods

After setting-up the scenario we found the solutions for each crew event using two methods. In the first method we used a real user from the AOCC, with the current tools available, to find the solutions. The user uses software that shows the roster of each crew member in a Gantt chart for a specific period. The user can scroll down the information, filter according to the crew rank and base, and sort the information by name, month duty, etc. Each user has a specific way of trying to find the solutions. However, we have observed that, in general, they follow these steps:

1. Open the roster for a one month period, starting two days before the current day. For example, let's suppose that the current day is 7th of June of 2006, they open the roster from the 5th of June until the 4th of July.
2. Filter the roster by crew rank and base, where the base is equal to the base where the crew event happened and crew rank is equal to the crew member rank that did not report for duty.
3. Order the information by month duty, in an ascendant order and by seniority in a descendent order.
4. Visually, they scroll down the information until they found a crew member with an open space for the period of time that corresponds to the duty to be assigned. This period of time takes into consideration the start and end time of the duty and also the time required for resting (ready date time).
5. If they do not found a crew member in the base specified, they try to find it in another base, filtering the information accordingly.
6. They assign the duty to the first crew member they found.

The data collected using this method is presented in Table 29. We point out that the data in columns marked with an asterisk where calculated manually, according to the formulas in the algorithm presented in Table 24 in chapter 2.2.5. Crew Recovery Sub-Organization. The reason for this is that the information system that is available for the users does not include information related with any kind of payroll.

	Duty ID	Base ID	Crew Grp	Rank	Crew Nr.	Crew Name	Sen.	Duty Pay (*)	Perdiem Pay (*)	Cost (*)
1	1ORY149S	A	2	CAB	229	Monica	576	0,00	72,00	72,00
2	1ORY149S	B	2	CAB	241	Michael	743	0,00	72,00	86,40
3	1ORY85P	A	1	CPT	213	Oscar M	189	942,90	106,00	1048,90
4	2LIS24X	A	2	CAB	242	Claudia	868	939,00	144,00	1083,00
5	3LIS25X	B	2	CAB	231	Mark	596	0,00	72,00	86,40
6	2LHR63P	B	1	CPT	171	Eduard C	230	777,00	212,00	1186,80
7	2LHR63P	B	1	OPT	273	House	192	0,00	148,00	177,60
8	1LHR31P	A	2	CCB	101	Arnold S	232	687,65	72,00	759,65
9	2LHR19P	B	2	CCB	184	Cristine H	391	0,00	144,00	172,80
10	1ZRH12X	C	1	CPT	250	John W	322	0,00	212,00	296,80
11	1LIS16S	A	2	CAB	230	Mary	592	0,00	72,00	72,00
12	1LIS16S	C	2	CAB	228	Angel	591	0,00	72,00	100,80
13	1LIS158T	B	2	CAB	249	Peter	855	0,00	72,00	86,40
14	3LIS174S	A	2	CAB	209	Allan	362	1051,60	216,00	1267,60
15	4LIS50A	A	1	OPT	274	Paul S	200	246,40	296,00	542,40
	Totals							4644,55	1982,00	7039,55

Table 29 - Crew Solution found for each Crew Event using method 1

In the second method we have used the sub-organization Crew recovery of our MAS as indicated in chapter 2.2.5. Crew Recovery Sub-Organization. The data collected is presented in Table 30.

	Duty ID	Base ID	Crew Grp	Rank	Crew Nr.	Crew Name	Sen.	Duty Pay	Perdiem Pay	Cost
1	1ORY149S	A	2	CAB	229	Monica	576	0,00	72,00	72,00
2	1ORY149S	B	2	CAB	241	Michael	743	0,00	72,00	86,40
3	1ORY85P	B	1	CPT	301	Marta	322	0,00	106,00	127,20
4	2LIS24X	C	2	CAB	280	Sophie	780	563,40	62,00	875,56
5	3LIS25X	B	2	CAB	231	Mark	596	0,00	72,00	86,40
6	2LHR63P	C	1	CPT	303	Clark	341	0,00	212,00	296,80
7	2LHR63P	A	1	OPT	320	Jarod	234	0,00	144,00	144,00
8	1LHR31P	B	2	CCB	290	Cecile	397	229,17	72,00	361,40
9	2LHR19P	B	2	CCB	184	Cristine H	391	0,00	144,00	172,80
10	1ZRH12X	C	1	CPT	250	John W	322	0,00	212,00	296,80
11	1LIS16S	A	2	CAB	230	Mary	592	0,00	72,00	72,00
12	1LIS16S	C	2	CAB	228	Angel	591	0,00	72,00	100,80
13	1LIS158T	B	2	CAB	249	Peter	855	0,00	72,00	86,40
14	3LIS174S	C	2	CAB	275	Parker	790	411,00	93,00	705,60
15	4LIS50A	B	1	OPT	321	Broots	240	0,00	296,00	355,20
	Totals							1203,57	1773,00	3839,36

Table 30 - Crew Solution found for each Crew Event using method 2

Table 31 shows a comparison of the results obtain through the above methods.

	Method 1		Method 2		Method 1/Method 2
	Total	%	Total	%	%
Base of the solution:					
- From the crew event base (A)	7	47%	3	20%	-57,14%
- From base B	6	40%	7	47%	16,67%
- From base C	2	13%	5	33%	150,00%
Time to Find Solution (avr sec)	101	100,00%	25	24,75%	-75,25%
- Base A (avr)	88	21%	24	24%	-72,73%
- Base B (avr)	110	27%	24	24%	-78,18%
- Base C (avr)	115	28%	26	26%	-77,39%
Total Costs:	7039,60	100,00%	3839,36	54,54%	-45,46%
Total Costs by Rank:					
- Rank CPT	2532,50	35,98%	720,80	18,77%	-71,54%
- Rank OPT	720,00	10,23%	499,20	13,00%	-30,67%
- Rank CCB	932,50	13,25%	534,20	13,91%	-42,71%
- Rank CAB	2854,60	40,55%	2085,16	54,31%	-26,95%
Total Costs by Base:					
- Base A	4845,55	92,42%	288,00	11,23%	-94,06%
- Base B	1796,40	34,26%	1275,80	49,77%	-28,98%
- Base C	397,60	7,58%	2275,56	88,77%	472,32%
Total Perdiem Pay:	1982,00	100%	1773,00	89,46%	-10,54%
Total Perdiem Pay by Rank:					
- Rank CPT	530,00	26,74%	530,00	29,89%	0,00%
- Rank OPT	444,00	22,40%	440,00	24,82%	-0,90%
- Rank CCB	216,00	10,90%	216,00	12,18%	0,00%
- Rank CAB	792,00	39,96%	587,00	33,11%	-25,88%
Total Perdiem Pay by Base:					
- Base A	978,00	77,50%	288,00	30,67%	-70,55%
- Base B	720,00	57,05%	834,00	88,82%	15,83%
- Base C	284,00	22,50%	651,00	69,33%	129,23%
Total Duty Pay:	4644,55	100%	1203,57	25,91%	-74,09%
Total Duty Pay by Rank:					
- Rank CPT	1719,90	37,03%	0,00	0,00%	-100,00%
- Rank OPT	246,40	5,31%	0,00	0,00%	-100,00%
- Rank CCB	687,65	14,81%	229,17	19,04%	-66,67%
- Rank CAB	1990,60	42,86%	974,40	80,96%	-51,05%
Total Duty Pay by Base:					
- Base A	3867,55	100,00%	0,00	0,00%	-100,00%
- Base B	777,00	20,09%	229,17	23,52%	-70,51%
- Base C	0,00	0,00%	974,40	100,00%	100,00%

Table 31 - Method 1 and Method 2 compared

From the results obtained we can see that in average, the second method took 25 seconds to find a solution and the first method took 101 seconds. Regarding the costs, the second method has a total cost of 3839.36 and the first method 7039.60. The second method is, in average 4 times faster than the first method in finding a solution and produces solutions that represent a decrease of 45.46% on the costs. From this data we can infer that our hypothesis was accepted. In the next section we present our interpretation of the results.

4. Conclusions

In this section we present the interpretation of the results which lead to our main contributions.

The focus of this thesis was in the design of a MAS for monitoring and operations recovery of an airline operations control centre. To accomplish this work we followed the principles of software engineering, more specifically, agent-oriented software engineering, starting from the requirements and finished with the implementation (in this case, partial implementation) of the designed system (and not the opposite as sometimes happens). That is, the system implemented here is the result of the modeling process followed from the beginning as well as incorporating in each of the following phases, what we have learnt from the previous ones. It is important to stress out that there was a clear rationale behind the process we have used to do this work.

We used GAIA (23), a software agent's methodology still under development, as the main methodology and complement it with TROPOS (18) to have a goal-oriented requirements analysis. We also think that the representations we have done using UML 2.0 of some of the deliverables of Gaia (for example, the organization structure, role and interaction diagrams and agent and service model) including the mappings between the abstractions used in the methodology and the UML concepts, are an added value of our work.

Starting from the observations of the AOCC of TAP Portugal, we have enumerated six hypothesis and predictions. For each one of them we also enumerate the expected results. The solutions and results we found out are the following:

Hypothesis one

Regarding this hypothesis we expected to obtain three things, represented in a formal way by the following expressions:

$$\begin{aligned} &Time(SolutionMAS(x)) \leq Time(SolutionTAP(x)) \\ &Use(LocalRes(SolutionMAS(x))) \geq Use(LocalRes(SolutionTAP(x))) \\ &Cooperation(SolutionMAS(x)) > Cooperation(SolutionTAP(x)) \end{aligned}$$

We did not evaluate our MAS regarding the solution of the three types of problems for which the system was designed, specifically, crew, aircraft and passenger recovery problems. However, taking into consideration the evaluation we have done for the crew recovery problems (see Table 31 in chapter 3. Results) we can conclude the following:

1. Our method is faster than the present method in TAP:

$$Time(SolutionMAS(25s)) \leq Time(SolutionTAP(101s))$$

This expression evaluates to true.

2. Our method does not use more local resources than the present method in TAP:

$$Use(LocalRes(SolutionMAS(20\%))) \geq Use(LocalRes(SolutionTAP(47\%)))$$

This expression evaluates to false.

3. Our method promotes an increase in cooperation from different operational bases:

$$Cooperation(SolutionMAS(BaseB(47\%))) > Cooperation(SolutionTAP(BaseB(40\%)))$$

$$Cooperation(SolutionMAS(BaseC(33\%))) > Cooperation(SolutionTAP(BaseC(13\%)))$$

This expressions evaluates to true.

Regarding 1) and 3) we accepted our hypothesis. The use of a computerized system to find and evaluate the solutions is the reason for our method to be faster than the present, and almost totally manual, method in TAP.

The reason for having an increase in the cooperation between different operational bases is that we evaluate all the solutions found and choose the one with less cost. In present method in TAP, they choose the first one they find, usually from the same base where the event was triggered. This cooperation is also possible to be inferred from the costs by base. In Table 31 is possible to see that the costs of base C had an increase of 472.32% while base A and base B decreased 94.06% and 28.98%, respectively. This means that our method used more resources from other bases than the base where the problem happened (base A).

Regarding 2) our hypothesis was rejected. We think that the reason for not having an increase in the use of local resources is related with the increase of the cooperation from different operational bases. These two indicators are inversely related and the total cost of the solution is the criteria that make the difference. This expression is not true only when the local resources are always less expensive than the non-local resources. From the information we collected this is a very unlikely hypothesis. The crew members, even from the same rank, have different salaries and consequently, it will affect the final cost of the solution.

Hypothesis two

Regarding this hypothesis we expected to obtain a considerable decrease in the costs of the solutions, expressed by the following expression:

$$Cost(SolutionMAS(x)) \leq Cost(SolutionTAP(x))$$

Subject to:

$$Effectiveness(SolutionMAS(x)) = Effectiveness(SolutionTAP(x))$$

Considering the evaluation we have done in chapter 3. Results, specifically the information in Table 31, we can conclude that our MAS produce much less expensive solutions:

$$Cost(SolutionMAS(3839,36)) \leq Cost(SolutionTAP(7039,60))$$

This was a decrease of 45.46%. Our hypothesis was accepted. Of course that we cannot infer that our MAS will always produce solutions that cost 45.46% less. It is not even possible to say that, in average, this decrease is valid. For that we need to evaluate much more situations, in different times of the year (we might have seasonal behaviors) and, then, find an average value. However, taking into consideration that our method includes information that is not available to the present method in TAP and assuming that the effectiveness of both methods are equal, we can state that our method will never produce more expensive solutions.

Hypothesis three

Regarding this hypothesis we believe that the Agent Model and the Service Model as presented in chapter 2.1.4.1. Agent Model and chapter 2.1.4.2. Service Model, respectively, will comply with the expected results, resulting in a system that:

1. Performs automatically the more repetitive tasks.
2. Gathers faster and more complete information.

3. Allows the human supervisors to take the final decision based on a more complete information.
4. Takes advantage of the distributed resources and of the distributed architecture of the system.
5. Allows scalability.

To be able to say if we accept or not our hypothesis, it would be necessary to fully implement the system and try it in a real situation, something that we could not do it during the time we wrote this thesis.

Regarding hypothesis four, five and six we were not able, in the time period available for this thesis, to evaluate the expected results. It would be necessary to fully implement the system. We hope to do it in a near future.

Besides the hypothesis that we have tested we think that this work brings up to the community of developers who use or want to use Gaia and JADE, the possibility of seeing how we overcame the difficulties and how we complement some missing parts of the methodology. This is the case, for example, for the lack of a process for gathering and modeling requirements. As stated previously, we also think that the representations we have done using UML 2.0 of some of the deliverables of Gaia (for example, the organization structure, role and interaction diagrams and agent and service model) including the mappings between the abstractions used in methodologies and the UML concepts, are an added value on MAS modeling that our work proposes to the MAS R&D community. However and to avoid that a designer has to produce too much documentation and perhaps, duplicate documentation, we would like to clearly define which of the models of GAIA could be replaced by our diagrams and which ones may be used jointly:

Replaced

- The table notation for the protocol definition (for example, Table 8), either in the preliminary or final interaction model, can be replaced by UML 2.0 Interaction Diagrams (Figure 15) for the same phases.
- The formal notation representing the organizational structure (Table 12) can be replaced by the UML 2.0 representation (for example, Figure 14).
- The table representation of the agent model (Table 18) can be replaced by a UML 2.0 class diagram as in Figure 19.
- The table representation of the service model (Table 19) can be replaced by a UML 2.0 class diagram as in Figure 20.

Used jointly:

- The combined graphical representation that includes the environment model of the preliminary role diagram (Figure 9), preliminary interactions diagram (Figure 10) and organization structure (Figure 12) can be used as a complement of the GAIA preliminary role model, preliminary interaction model and organization structure UML 2.0 representation (Figure 14), respectively.
- The UML 2.0 representation of the role and interaction model (for example, Figure 17) can help to better visualize the organization with their roles, activities and protocols.

As a final contribution of this work to the community we would like to emphasize the possibility of following the development of a real-world application, from requirements gathering to implementation, and perceive the rationale that was behind that development.

Limitations and Future Works

We know that our proposal requires a huge effort in dedication and time and that we did not do everything in this thesis. However, we believe that this work creates the necessary foundation and test workbench for improvements in this domain.

As it is possible to see from the previous chapters, our work has some limitations. We would like to point out the following:

- ✓ We assume that the environment has the pairing already created or changed according to the needs of the problem. Our MAS only assigns crew members to existing pairings and does not change any pairing as part of the solution. In a real situation this is a limitation. We have seen from the AOCC in TAP that, quite often, they need to split the pairings so that a crew member can be assigned to it.
- ✓ Our MAS does not include a rule engine. This business domain has several rules, applied to the scheduling of crew members, and the lack of a rule engine is a drawback.

For the future we intend to propose the following tasks:

- ✓ To completely finish the MAS, not only to be used in a specific airline company but, also, to be used as a tool in the laboratory to experiment new ideas regarding new or improved algorithms/solutions. This can be algorithms/solutions for crew, aircraft and passenger recovery, learning and applying the knowledge in future rosters, negotiation, etc.
- ✓ To include the crew pairing problem as part of a more integrated solution. The idea is to also solve automatically the pairings that will be necessary to solve a crew/aircraft problem, including commercial changes.
- ✓ To include an inference engine, like JESS (42), for example, to create and validate all rules related with this business domain.
- ✓ To access through Web Services to the information that exists in a database localized in a different geographical place. This way we will have a new layer of abstraction so that each base can access the information of the operational base resources.
- ✓ To define an ontology, specific to this business domain that might help specially when using the Electronic Market for cooperation between different airlines.
- ✓ To connect our MAS to the Electronic Market of solutions, as specified in (43). This would contribute to increase the cooperation between different airlines in solving this kind of problems.

We also would like to define several types of organization structures for this MAS and, then, perform the necessary tasks to measure and evaluate the organization structure according to the defined method. The idea or hypothesis is to see if the type of Organization Structure defined for this kind of problems has any influence in the results, for example, did it choose better solutions (that is, with a less cost?), was it faster in finding the solutions? Did it find more possible solutions to choose from? This analysis should be based in the work of Mark S. Fox as presented in (40).

REFERENCES

1. *Airline crew rostering. problem types, modeling, and optimization.* **Kohl, Niklas and Karisch, Stefan E.** s.l. : Annals of Operations Research, 2004, Vol. 127, pp. 223-257.
2. **Kohl, Niklas, et al.** *Airline disruption management - perspectives, experiences and outlook.* s.l. : Carmen Research, 2004. Technology Report CRTR-0407.
3. **Wooldridge, Michael.** When is an Agent-Based Solution Appropriate? *An Introduction to Multiagent Systems.* West Sussex, England : John Wiley & Sons, Ltd., 2002, pp. 225-226.
4. **The Intelligent Software Agents Lab.** Multi-Agent Systems. *Intelligent Software Agents Lab.* [Online] The Robotics Institute - Carnegie Mellon University, 2001. [Cited: 11 29, 2006.] <http://www.cs.cmu.edu/~softagents/multi.html>.
5. **TAP Portugal.** TAP Portugal Web Site. *Flytap.* [Online] TAP Portugal. [Cited: 11 29, 2006.] <http://www.flytap.pt>.
6. *Applications of Operations Research in the Air Transport Industry.* **Barnhart, Cynthia, Belobaba, Peter and Odoni, Amedeo R.** 4, November 2003, Transportation Science, Vol. 37, pp. 368-391.
7. **Clausen, Jens, Larsen, Allan and Larsen, Jesper.** *Disruption Management in the Airline Industry - Concepts, Models and Methods.* Informatics and Mathematical Modelling, Technical University of Denmark. Denmark : DTU, 2005. Technology Report IMM-Technical Report-2005-01.
8. **Rosenberger, Jay M, Johnson, Ellis L and Nemhauser, George L.** *Rerouting Aircraft for Airline Recovery.* Georgia Institute of Technology. s.l. : Technical Report TLI-LEC 01-04, 2001.
9. *A Stochastic Model of Airline Operations.* **Rosenberger, Jay M, et al.** 4, 2002, Transportation Science, Vol. 36, pp. 357-377.
10. *A Proactive Crew Recovery Decision Support Tool for Commercial Airlines during Irregular Operations.* **Abdelgahny, Ahmed, et al.** 2004, Annals of Operations Research, Vol. 127, pp. 309-331.
11. *Flight Operations Recovery: New Approaches Considering Passenger Recovery.* **Bratu, Stephane and Barnhart, Cynthia.** 3, June 2006, Journal of Scheduling, Vol. 9, pp. 279-298.
12. **Ladislav, Letovsky.** *Airline Operations Recovery: An Optimization Approach.* Ph.D. Thesis. Atlanta, USA : Georgia Institute of Technology, 1997.
13. *Managing Flight Operations.* **Martins, Joao P and Morgado, Ernesto.** London : s.n., 1996. Proceedings of ATTIS'96.
14. **Airline Group of the International Federation of Operational Research Societies.** AGIFORS Online. *AGIFORS Web Site.* [Online] [Cited: December 16, 2006.] <http://www.agifors.org>.
15. **Digipede Technologies, LLC.** DIGIPEDE: Grid Computing for Windows. *DIGIPEDE Company Site.* [Online] Digipede Technologies, LLC. [Cited: 12 02, 2006.] <http://www.digipede.net/>.
16. *Providing Advanced, Personalised Infomobility Services using Agent Technology.* **Moraitis, Pavlos, Petraki, Eleftheria and Spanoudakis, Nikolaos I.** Cambridge, UK : s.n., 2003. 23rd SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence (AI2003). pp. 35-48.
17. *Agent Oriented Analysis using MESSAGE/UML.* **Caire, Giovanni, et al.** s.l. : Springer, 2001. Proceedings Agent Oriented Software Engineering (AOSE).

18. *Tropos: An Agent-Oriented Software Development Methodology*. **Bresciani, Paolo, et al.** 3, s.l. : Kluwer Academic Publishers, 2004, Autonomous Agents and Multi-Agent Systems, Vol. 8, pp. 203-236.
19. *Prometheus: A Methodology for Developing Intelligent Agents*. **Padgham, Lin and Winikoff, Michael.** Bologna, Italy : s.n., 2002. Proceedings of the 3rd International Workshop on Agent Oriented Software Engineering (AOSE 2002).
20. *An Overview of the Multi-Agent Systems Engineering Methodology*. **Wood, Mark F and DeLoach, Scott A.** Limerick, Ireland : Springer-Verlag, 2001. Proceedings of the Agent-Oriented Software Engineering 1st International Workshop (AOSE-2000). pp. 207-222.
21. *Developing Multi-Agent Systems with Agent Tool*. **DeLoach, Scott A and Wood, Mark F.** Boston, Massachusetts, USA : Springer-Verlag, 2001. Intelligent Agents VII: agent Theories Architectures and Languages 7th International Workshop (ATAL 2000). Vol. LNCS 1986, pp. 46-60.
22. *A CASE Tool Supported Methodology for the Design of Multi-Agent Systems*. **Cossentino, Massimo and Potts, Colin.** Las Vegas, Nevada, USA : s.n., 2002. Proceedings of the 2002 International Conference on Software Engineering Research and Practice (SERP02).
23. *Developing Multiagent Systems: The Gaia Methodology*. **Zambonelli, Franco, Jennings, Nicholas R and Wooldridge, Michael.** 3, s.l. : ACM Inc., 2003, ACM Transactions on Software Engineering and Methodology, Vol. 12, pp. 317-370.
24. **Bergenti, Federico, Gleizes, Marie-Pierre and Zambonelli, Franco, [ed.].** *Methodologies and Software Engineering for Agent Systems: The Agent-oriented Software Engineering Handbook*. s.l. : Kluwer Academic Publishers, 2004. ISBN 1402080573.
25. **Perini, Anna, et al.** Tool for Agent Oriented Modeling (TAOM4E). [Online] 08 25, 2006. [Cited: 12 02, 2006.] <http://sra.itc.it/tools/taom4e/>.
26. **DeLoach, Scott A.** The agentTool Project. *Multiagent & Cooperative Robotics Laboratory*. [Online] [Cited: December 26, 2006.] <http://macr.cis.ksu.edu/projects/agentTool/agentool.htm>.
27. **Moraitis, Pavlos, Petraki, Eleftheria and Spanoudakis, Nikolaos I.** Engineering JADE Agents with the GAIA Methodology. [ed.] R Kowalszyk, et al. *Agent Technologies, Infrastructures, Tools and Applications for E-Services*. Berlin : Springer-Verlag, 2003, Vol. LNAI 2592, pp. 77-91.
28. **Cossentino, Massimo, et al.** Introducing Pattern Reuse in the Design of Multi-Agent Systems. [ed.] R Kowalszyk, et al. *Agent Technologies, Infrastructures, Tools and Applications for E-Services*. Berlin : Springer-Verlag, 2003, Vol. LNAI 2592, pp. 107-120.
29. *Agent Oriented Software Engineering with INGENIAS*. **Gómez-Sanz, Jorge and Pavón, Juan.** s.l. : Berlin: Springer-Verlag, 2003. Multi-Agent Systems and Applications III Proceedings of the 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2003). Vol. LNCS 2691, pp. 394-403.
30. **Bellifemine, Fabio, et al.** JADE 3.3 Programmer's Guide. s.l. : TILab S.p.A., 2004.
31. *The GAIJA2JADE Process for Multi-Agent Systems Development*. **Moraitis, Pavlos and Spanoudakis, Nikolaos.** 2-4, 2006, Applied Artificial Intelligence, Vol. 20, pp. 251-273.

-
32. **Object Management Group, Inc.** The Object Management Group (OMG). *The Object Management Group*. [Online] 1997-2006. [Cited: 12 01, 2006.] <http://www.omg.org/>.
 33. *Software Development with Objects, Agents, and Services*. **Huhns, Michael N.** Vancouver, Canada : s.n., 2004. Proceedings of the 3rd International Workshop on Agent-Oriented Methodologies.
 34. *From Object-Oriented to Goal-Oriented Requirements Analysis*. **Mylopoulos, John, Chung, Lawrence and Yu, Eric.** 1, 01 1999, Communications of the ACM, Vol. 42, pp. 31-37.
 35. *UML 2.0 and Agents: How to Build Agent-Based Systems with the New UML Standard*. **Bauer, Bernhard e Odell, James.** 2, 2005, Journal of Engineering Applications of Artificial Intelligence, Vol. 18, pp. 141-157.
 36. **IBM Corporation.** IBM Rational Software. *IBM*. [Online] [Cited: 12 02, 2006.] <http://www-306.ibm.com/software/rational/>.
 37. —. DB2 Product family. *IBM*. [Online] [Cited: 12 02, 2006.] <http://www-306.ibm.com/software/data/db2/>.
 38. **TAP Portugal.** Part A - 01.03. *Airline Operations Police Manual (AOPM)*. s.l. : TAP Portugal , 12 15, 2004.
 39. *Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems*. **Zambonelli, Franco, Jennings, Nicholas R and Wooldridge, Michael.** 3, 2001, International Journal of Software Engineering and Knowledge Engineering, Vol. 11, pp. 303-328.
 40. *An Organizational View of Distributed Systems*. **Fox, Mark S.** 3, January 1981, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 11, pp. 70-80.
 41. *Agent UML: A Formalism for Specifying Multiagent Software Systems*. **Bauer, Bernhard, Muller, Jorg P and Odell, James.** 3, April 2001, International Journal of Software Engineering and Knowledge Engineering, Vol. 11, pp. 207-230.
 42. **Sandia National Laboratories.** Jess, The Rule Engine for Real Programmers. *Jess, The Rule Engine*. [Online] Sandia National Laboratories, October 31, 2006. [Cited: December 05, 2006.] <http://www.jessrules.com/>.
 43. *Crew and Aircraft Recovery Through a Multi-Agent Airline Electronic Market*. **Malucelli, Andreia, Castro, Antonio and Oliveira, Eugenio.** [ed.] Sandeep Krishnamurthy and Pedro Isaias. Barcelona, Spain : IADIS Press, 2006. Proceedings of IADIS International Conference e-Commerce 2006. pp. 51-58. ISBN: 972-8924-23-2.

ANNEX A - BIBLIOGRAPHY

JADE Tutorial – JADE Programming for Beginners.

04/12/2003

Author: Giovanni Caire (TILAB, formerly CSELT)

JADE Programmer's Guide

22/11/2005

Authors: Fabio Bellifemine, Giovanni Caire, Tiziana Trucco (TILAB, formerly CSELT), and Giovanni Rimassa (University of Parma).

SC00037J - FIPA Communicative Act Library Specification

03/12/2002

Author: Foundation for Intelligent Physical Agents.

JADE Tutorial and Primer

URL (available as of June 2006):

<http://www.iro.umontreal.ca/~vaucher/Agents/Jade/JadePrimer.html>

Authors: Jean Vaucher and Ambroise Ncho

Informatics Department, Montreal University

UML 2.0 in a Nutshell

30/06/2005

Authors: Dan Pilone and Neil Pitman

Publisher: O'Reilly ISBN-10: 0596007957 ISBN-13: 978-0596007959

ANNEX B – PRELIMINARY ROLE MODEL

Role Schema: CrewFind
Description: This preliminary role tries to find the best crew member to be assigned to a flight, after a request received by the organization (RosterCrewMonitor). It should use different techniques (that is, different algorithms) to find possible solutions and inform the organization about them.
Protocols and Activities: <u>InitCrewCFP</u> , <u>ChooseBestCrew</u>
Permissions: reads Roster // to obtain available crew and/or crew from the stand by roster. reads Pairings //to obtain alternative pairings and/or info about affected pairing.
Responsibilities
Liveness: CrewFind = (<u>InitCrewCFP</u> . <u>[ChooseBestCrew]</u>) ^w .
Safety: - successful_connection_with_Roster = true - number_of_crew_requests >= 1 // at least one crew request must exist - request_crew_status = open // processes only open requests - request_crew_status_after_find = close // if unable to find a crew member - request_crew_status_after_find = found // if a crew member was found

Table 8 – CrewFind preliminary role

Role Schema: CrewAssign
Description: This preliminary role assigns the crew member found by the CrewFind role. Before assigning the crew members it needs to check if the environment is still the same when the problem was detected and/or when the solution was found. If the environment has changed in a way that does not allow to apply the solution (or makes the solution unnecessary) it should inform the organization accordingly. After applying successfully the solution should also inform the organization.
Protocols and Activities: <u>CheckCrewEnvironment</u> , <u>AssignCrew</u>
Permissions: create, update, delete Roster // to assign the crew member to the specific(s) pairing(s) create, update, delete Pairings // all the affects pairings
Responsibilities
Liveness: CrewAssign = (<u>CheckCrewEnvironment</u> . <u>AssignCrew</u>) ^w .
Safety: - successful_connection_with_Roster = true - successful_connection_with_Pairings = true - atomic_transactions = true // changes must be atomic - number_of_crew_solutions >= 1 // at least one crew solution should exist - request_crew_status = found // processes only found crew solutions requests - request_crew_status_after_assign = close

Table 9 – CrewAssign preliminary role

Role Schema: RosterAircraftMonitor
Description: This preliminary role involves monitoring the aircraft roster and the airports information system, for events related with flight delays. After detecting one of these events it will request the organization a solution. It should be able to trace previous requests, avoiding duplicates, until receive a message regarding the status of the request.
Protocols and Activities: <u>CheckForNewACEvents</u> , <u>UpdateACEventStatus</u>
Permissions: reads <u>AircRoster</u> // to obtain information regarding delayed flights.
Responsibilities
Liveness: RosterAircraftMonitor = (<u>CheckForNewACEvents</u>) ^w (<u>UpdateACEventStatus</u>) ^w .
Safety: - successful_connection_with_AirpInfoSys = true - successful_connection=with_AircRoster = true - new_aircraft_request <> existing_unclosed_aircraft_request

Table 10 – RosterAircraftMonitor preliminary role

Role Schema: AircraftFind
Description: This preliminary role tries to find the best solution for a flight delay, when it is related with aircraft problems. It should use different techniques (that is different algorithms) to find possible solutions and inform the organization about them.
Protocols and Activities: <u>InitAcCFP</u> , <u>ChooseBestAC</u>
Permissions: reads <u>AircRoster</u> // to obtain alternative aircrafts.
Responsibilities
Liveness: AircraftFind = (<u>InitAcCFP</u> . <u>ChooseBestAC</u>) ^w .
Safety: - successful_connection_with_AircRoster = true - number_of_aircraft_requests >= 1 // at least one aircraft request must exist - request_aircraft_status = open // processes only open requests - request_aircraft_status_after_find = close // if unable to find a solution - request_aircraft_status_after_find = found // if a solution was found

Table 11 – AircraftFind preliminary role

Role Schema: AircraftAssign
Description: This preliminary role assigns the aircraft found by the AircraftFind role. Before assigning the aircraft it needs to check if the environment is still the same when the problem was detected and/or when the solution was found. If the environment has changed in a way that does not allow to apply the solution (or makes the solution unnecessary) it should inform the organization accordingly. After applying successfully the solution should also inform the organization.
Protocols and Activities: <u>CheckACEnvironment</u> , <u>AssignAC</u>
Permissions: create, update, delete AirRoster // to assign different flights to the aircrafts
Responsibilities
Liveness: AircraftAssign = (<u>CheckACEnvironment.AssignAC</u>) ^w .
Safety: - successful_connection_with_AirRoster = true - number_of_aircraft_solutions >= 1 // at least one aircraft solution should exist - request_aircraft_status = found // processes only found aircraft solutions requests - request_aircraft_status_after_assign = close

Table 12 – AircraftAssign preliminary role

Role Schema: PaxMonitor
Description: This preliminary role involves monitoring the passenger check-in for events related with passengers confirmed or already checked-in affected by flight delays and for passengers affected by overbooking situations. After detecting one of these events it will request to the organization a solution. It should be able to trace previous requests, avoiding duplicates, until receive a message regarding the status of the request
Protocols and Activities: <u>CheckForNewPaxEvents</u> , <u>UpdatePaxEventStatus</u>
Permissions: reads ResInfoSys // to obtain check-in status.
Responsibilities
Liveness: PaxMonitor = (<u>CheckForNewPaxEvents</u>) ^w (<u>UpdatePaxEventStatus</u>) ^w .
Safety: - successful_connection_with_ResInfoSys = true - new_pax_request \diamond existing_unclosed_pax_request

Table 13 – PaxMonitor preliminary role

Role Schema: PaxFind
Description: This preliminary role tries to find the best solution for the events detected and broadcasted by the PaxMonitor role. It should suggest a plan to be followed by the company, including checking-in in others flights (even from others companies) and send passengers to the hotel if appropriate. It should inform the user with role PaxApply of the solution found (a plan).
Protocols and Activities: <u>FindPaxSolution</u>
Permissions: reads ResInfoSys // to obtain check-in status and flight availability reads AirInfoSys // to obtain flight information.
Responsibilities Liveness: PaxFind = (<u>FindPaxSolution</u>).
Safety: - successful_connection_with_ResInfoSys = true - number_of_pax_requests >= 1 // at least one pax request must exist - request_pax_status = open // processes only open requests - request_pax_status_after_find = close // if unable to find a pax solution - request_pax_status_after_find = found // if a pax solution was found

Table 14 – PaxFind preliminary role

Role Schema: PaxApply
Description: This preliminary role applies the solution plan found by the PaxFind role. It does not deal directly with computer systems. Usually a human operator will perform the tasks recommended by the solution plan. Before applying the plan should check if the environment is still the same when the problem was detected and/or when the solution was found. If the environment has changed in a way that does not allow to apply the solution (or makes the solution unnecessary) it should inform the organization accordingly. After applying successfully the solution should also inform the organization.
Protocols and Activities: (To be fully determined in Architectural Design)
Permissions: create, update, delete ResInfoSys // to do the check-in for the passengers
Responsibilities Liveness: PaxApply = (<u>CheckPaxEnvironment</u> , <u>AssignPax</u>) ^w .
Safety: - successful_connection_with_ResInfoSys = true - number_of_pax_solutions >= 1 // at least one pax solution should exist - request_pax_status = found // processes only found pax solutions requests - request_pax_status_after_assign = close

Table 15 – PaxApply preliminary role

ANNEX C – PRELIMINARY INTERACTION MODEL

Protocol name: reportCrewEventStatus		
Initiator (role(s)): CrewFind	Partner (role(s)): RosterCrewMonitor	Input: Close if unable to find a crew member OR Found if a crew member was found; in this case it is waiting for the solution to be applied.
Description: After finishing the task of trying to find a crew member to the open position, it is necessary to inform the status of the request to the Roster Crew Monitor.		Output: OK if able to update the status of the original request OR NOT OK if unable to update the status

Table 32 – Preliminary protocol definition for reportCrewEventStatus

Protocol name: applyCrewSolution		
Initiator (role(s)): CrewFind	Partner (role(s)): CrewAssign	Input: Crew solution information
Description: When a crew member was found (crew solution) it is necessary to apply the solution. This protocol initiates the request for applying the solution to the CrewAssign role.		Output: Yes; I will apply the solution (means the environment conditions remains) OR No; I cannot apply the solution (environment changed).

Table 33 – Preliminary protocol definition for applyCrewSolution

Protocol name: reportCrewSolutionStatus		
Initiator (role(s)): CrewAssign	Partner (role(s)): CrewFind RosterCrewMonitor	Input: Close request
Description: After applying the solution it is necessary to inform the status of the requests. This protocol enables to share that information with both roles.		Output: OK if able to update the status of the original request OR NOT OK if unable to update the status

Table 34 – Preliminary protocol definition for reportCrewSolutionStatus

Protocol name: informACEvents		
Initiator (role(s)): RosterAircraftMonitor	Partner (role(s)): AircraftFind	Input: Flight Delay Information
Description: After a flight delay has been detected it is necessary to find a solution to the problem. For that it is necessary to send details about the problem (technical problem, ATC problem, weather problem, etc.) for a solution to be found.		Output: Yes; I will try to find a solution OR No; I cannot process the request (see safety conditions on AircraftFind role).

Table 35 – Preliminary protocol definition for informACEvents

Protocol name: reportACEventStatus		
Initiator (role(s)): AircraftFind	Partner (role(s)): RosterAircraftMonitor	Input: Close if unable to find a solution OR Found if a solution was found; in this case it is waiting for the solution to be applied.
Description: After finishing the task of trying to find a solution to the problem, it is necessary to inform the status of the request to the Aircraft Crew Monitor.		Output: OK if able to update the status of the original request OR NOT OK if unable to update the status

Table 36 - Preliminary protocol definition for reportACEventStatus

Protocol name: applyAircraftSolution		
Initiator (role(s)): AircraftFind	Partner (role(s)): AircraftAssign	Input: Aircraft solution information
Description: When an aircraft was found (aircraft solution) it is necessary to apply the solution. This protocol initiates the request for applying the solution to the AircraftAssign role.		Output: Yes; I will apply the solution (means the environment conditions remains) OR No; I cannot apply the solution (environment changed).

Table 37 - Preliminary protocol definition for applyAircraftSolution

Protocol name: reportAircraftSolutionStatus		
Initiator (role(s)): AircraftAssign	Partner (role(s)): AircraftFind RosterAircraftMonitor	Input: Close request
Description: After applying the solution it is necessary to inform the status of the requests. This protocol enables to share that information with both roles.		Output: OK if able to update the status of the original request OR NOT OK if unable to update the status

Table 38 - Preliminary protocol definition for reportAircraftSolutionStatus

Protocol name: informPaxEvents		
Initiator (role(s)): PaxMonitor	Partner (role(s)): PaxFind	Input: Pax problem information
Description: After an event has been detected (pax affected by delays and/or overbooking) it is necessary to find a solution. For that it is necessary to send details about the pax problem so that a solution might be found, usually in the form of a suggested plan to be followed by the company.		Output: Yes; I will try to find a solution OR No; I cannot process the request (see safety conditions on CrewFind role).

Table 39 - Preliminary protocol definition for informPaxEvents

Protocol name: reportPaxEventStatus		
Initiator (role(s)): PaxFind	Partner (role(s)): PaxMonitor	Input: Close if unable to find a solution OR Found if a plan was found; in this case it is waiting for the solution to be applied.
Description: After finishing the task of trying to a plan to be followed, it is necessary to inform the status of the request to the Pax Monitor.		Output: OK if able to update the status of the original request OR NOT OK if unable to update the status

Table 40 - Preliminary protocol definition for Report paxRequestStatus

Protocol name: applyPaxSolution		
Initiator (role(s)): PaxFind	Partner (role(s)): PaxApply	Input: Plan information
Description: When a plan was found (pax solution) it is necessary to apply the solution. This protocol initiates the request for applying the solution to the PaxApply role.		Output: Yes; I will apply the solution (means the environment conditions remains) OR No; I cannot

	apply the solution (environment changed).
--	--

Table 41 - Preliminary protocol definition for applyPaxSolution

Protocol name: reportPaxSolutionStatus		
Initiator (role(s)): PaxApply	Partner (role(s)): PaxFind PaxMonitor	Input: Close request
Description: After applying the solution it is necessary to inform the status of the requests. This protocol enables to share that information with both roles.		Output: OK if able to update the status of the original request OR NOT OK if unable to update the status

Table 42 - Preliminary protocol definition for reportPaxSolutionStatus

ANNEX D – ROLE MODEL

Role Schema: CrewFind (CF)
Description: Tries to find the best crew member to be assigned to a flight, after receive crew event information. Controls other roles specialized in different techniques to find possible solutions. Sends CFP to the roles requesting solutions (through a Contract NET protocol), chooses the best solution according to a utility and informs role Operations Schedule Manager.
Protocols and Activities: <u>InitCrewCFP</u> , <u>ChooseBestCrew</u> , informsCrewEvent, requestCrewSolution, sendsCrewSolution, reportCrewEventStatus
Permissions:
Responsibilities
Liveness: CrewFind = (informsCrewEvent ^W . <u>InitCrewCFP</u> .requestCrewSolution. [<u>ChooseBestCrew</u> .sendsCrewSolution]. reportCrewEventStatus) ^W
Safety: - number_of_crew_requests >= 1 // at least one crew request must exist - request_crew_status = open // processes only open requests - request_crew_status_after_find = close // if unable to find a crew member - request_crew_status_after_find = found // if a crew member was found

Table 43 – CrewFind (CF) role

Role Schema: TapCRHeuristic (TapCR)
Description: Tries to find a crew member (or a list of crew members) according to the CFP received (Contract NET protocol), implementing a known heuristic used by human operators in TAP.
Protocols and Activities: <u>FindCrewHeuristic</u> , requestCrewSolution
Permissions: reads Roster // to obtain available crew and/or crew from the stand by roster. reads Pairings //to obtain alternative pairings and/or info about affected pairing.
Responsibilities
Liveness: TapCRHeuristic = (requestCrewSolution. <u>FindCrewHeuristic</u> .requestCrewSolution) ^W
Safety: - successful_connection_with_Roster = true - successful_connection_with_Pairings = true

Table 44 - TapCRHeuristic (TapCR) role

Role Schema: OtherCRAAlgorithm (OtherCR)
Description: Tries to find a crew member (or a list of crew members) according to the CFP received (Contract NET protocol), implementing a Crew Recovery algorithm that already exists.
Protocols and Activities: <u>FindCrewOther</u> , requestCrewSolution
Permissions: reads Roster // to obtain available crew and/or crew from the stand by roster. reads Pairings //to obtain alternative pairings and/or info about affected pairing.
Responsibilities Liveness: OtherCRAAlgorithm = (requestCrewSolution. <u>FindCrewOther</u> .requestCrewSolution) ^W
Safety: - successful_connection_with_Roster = true - successful_connection_with_Pairings = true

Table 45 - OtherCRAAlgorithm (OtherCR) role

Role Schema: OperationsScheduleManager (OSM)
Description: Represents the user in charge of controlling the application of a found crew solution. Receives the solution from the CrewFind and presents a detailed explanation of the solution to the user, requesting authorization to apply it. If the user authorizes, request the application of the solution to the Crew Assign. Reports the status of the request to the CrewFind.
Protocols and Activities: <u>ShowCrewSolution</u> , <u>AskManagerPermission</u> , <u>InfoManager</u> , sendsCrewSolution, applyCrewSolution, reportCrewSolutionStatus
Permissions:
Responsibilities Liveness: OperationsScheduleManager = (sendsCrewSolution ^W . <u>ShowCrewSolution</u> . <u>AskManagerPermission</u> . sendsCrewSolution. [applyCrewSolution.reportCrewSolutionStatus. <u>InfoManager</u>]) ^W
Safety: - number_of_crew_solutions >= 1 // at least one crew solution must exist - request_crew_status = found // processes only found crew solution requests

Table 46 – OperationsScheduleManager (OSM) Role

Role Schema: CrewAssign (CA)
Description: Applies the crew solution authorized by OperationsScheduleManager. Before applying the solution and, consequently, assigning the crew members checks if the environment is still the same when the problem was detected and/or when the solution was found. If the environment has changed in a way that does not allow applying the solution (or makes the solution unnecessary) informs the OperationsScheduleManager. After applying successfully the solution informs the organization.
Protocols and Activities: <u>CheckCrewEnvironment</u> , <u>AssignCrew</u> , applyCrewSolution, reportCrewSolutionStatus
Permissions: create, update, delete Roster // to assign crew member to the specific(s) pairing(s) create, update, delete Pairings // all the affects pairings
Responsibilities Liveness: CrewAssign = (applyCrewSolution ^W . <u>CheckCrewEnvironment</u> . [reportCrewSolutionStatus]. <u>AssignCrew</u> .reportCrewSolutionStatus) ^W
Safety: - successful_connection_with_Roster = true - successful_connection_with_Pairings = true - atomic_transactions = true // changes must be atomic - request_crew_status_after_assign = close

Table 47 – CrewAssign (CA) role

Role Schema: RosterAircraftMonitor (RAM)
Description: Monitors the aircraft roster for events related with flight delays. After detecting one of these events it will request to the organization a solution. Traces previous requests and avoids duplicates, until receive a message regarding the status of the request.
Protocols and Activities: <u>CheckForNewACEvents</u> , <u>UpdateACEventStatus</u> , informsACEvent, reportACEventStatus
Permissions: reads AircRoster // to obtain information regarding delayed flights. create, read, update AC Events Queue Data Class // keeps a record of events status
Responsibilities Liveness: RosterAircraftMonitor = (<u>CheckForNewACEvents</u> ^W .informsACEvent) ^W (reportACEventStatus ^W . <u>UpdateACEventStatus</u>) ^W
Safety: - successful_connection_with_AircRoster = true - successful_connection_with_ACEvents = true - new_aircraft_request <> existing_unclosed_aircraft_request

Table 48– RosterAircraftMonitor (RAM) role

Role Schema: AircraftFind (AF)
Description: Tries to find the best solution for a flight delay, after receive aircraft event information. Controls other roles specialized in different techniques to find possible solutions. Sends CFP to the roles requesting solutions (through a Contract NET protocol), chooses the best solution according to a utility and informs role Operational Control Supervisor.
Protocols and Activities: <u>InitAcCFP</u> , <u>ChooseBestAC</u> , informsACEvent, requestACSolution, sendsACSolution, reportACEventStatus
Permissions:
Responsibilities
Liveness: AircraftFind = (informsACEvent ^W . <u>InitAcCFP</u> .requestACSolution. [<u>ChooseBestAC</u> .sendsACSolution]. reportACEventStatus) ^W
Safety: - number_of_aircraft_requests >= 1 // at least one aircraft request must exist - request_aircraft_status = open // processes only open requests - request_aircraft_status_after_find = close // if unable to find a solution - request_aircraft_status_after_find = found // if a solution was found

Table 49 - AircraftFind (AF) role

Role Schema: TapARHeuristic (TapAR)
Description: Tries to find an aircraft solution (or a list of aircraft solutions) according to the CFP received (Contract NET protocol), implementing a known heuristic used by human operators in TAP.
Protocols and Activities: <u>FindACHeuristic</u> , requestACSolution
Permissions: reads AircRoster // to obtain information about the aircrafts roster.
Responsibilities
Liveness: TapARHeuristic = (requestACSolution. <u>FindACHeuristic</u> .requestACSolution) ^W
Safety: - successful_connection_with AircRoster = true

Table 50 - TapARHeuristic (TapAR) role

Role Schema: OtherARAlgorithm (OtherAR)
Description: Tries to find an aircraft solution (or a list of aircraft solutions) according to the CFP received (Contract NET protocol), implementing an Aircraft Recovery algorithm that already exists.
Protocols and Activities: <u>FindACOther</u> , requestACSolution
Permissions: reads <u>AircRoster</u> // to obtain information about the aircrafts roster.
Responsibilities
Liveness: OtherARAlgorithm = (requestARSolution. <u>FindAROther</u> .requestARSolution) ^W
Safety: - <u>successful_connection_with_AircRoster</u> = true

Table 51 - OtherARAlgorithm (OtherAR) role

Role Schema: OperationalControlSupervisor (OCS)
Description: Represents the user in charge of controlling the application of a found aircraft solution and of a passenger recovery solution. Receives the solution from the AircraftFind and PaxFind and presents a detailed explanation of the solution to the user, requesting authorization to apply it. If the user authorizes, request the application of the solution to the AircraftAssign and to the PaxApply, respectively. Reports the status of the request to the AircraftFind and to the PaxFind.
Protocols and Activities: <u>ShowACSolution</u> , <u>ShowPaxSolution</u> , <u>AskSupervisorPermission</u> , <u>Infosupervisor</u> , sendsACSolution, applyACSolution, reportACSolutionStatus, sendsPaxSolution, applyPaxSolution, reportPaxSolutionStatus
Permissions:
Responsibilities
Liveness: OperationalControlSupervisor = (sendsACSolution ^W sendsPaxSolution ^W). (<u>ShowACSolution</u> <u>ShowPaxSolution</u>). <u>AskSupervisorPermission</u> . (sendsACSolution sendsPaxSolution). ([<u>applyACSolution</u> .reportACSolutionStatus. <u>InfoSupervisor</u>] [<u>applyPaxSolution</u> .reportPaxSolutionStatus. <u>InfoSupervisor</u>])) ^W
Safety: - <u>number_of_aircraft_solutions</u> >= 1 // at least one aircraft solution must exist - <u>number_of_pax_solutions</u> >=1 //at least one aircraft solution must exist - <u>request_aircraft_status</u> = found // processes only found aircraft solution requests - <u>request_pax_status</u> = found // processes only found aircraft solution requests

Table 52– OperationalControlSupervisor (OCS) Role

Role Schema: AircraftAssign (AA)
<p>Description: Applies the aircraft solution authorized by OperationalControlSupervisor. Before applying the solution checks if the environment is still the same when the problem was detected and/or when the solution was found. If the environment has changed in a way that does not allow applying the solution (or makes the solution unnecessary) informs the OperationalControlSupervisor. After applying successfully the solution informs the organization.</p>
<p>Protocols and Activities: <u>CheckACEnvironment</u>, <u>AssignAC</u>, applyACSolution, reportACSolutionStatus</p>
<p>Permissions: create, update, delete AirRoster // to assign different flights to the aircrafts</p>
<p>Responsibilities Liveness: AircraftAssign = (applyACSolution^W.<u>CheckACEnvironment</u>. [reportACSolutionStatus].<u>AssignAC</u>.reportACSolutionStatus)^W</p> <p>Safety: - successful_connection_with_AirRoster = true - atomic_transactions = true // changes must be atomic - request_aircraft_status_after_assign = close</p>

Table 53 - AircraftAssign (AA) role

Role Schema: PaxMonitor (PM)
<p>Description: Monitors the passenger check-in for events related with passengers confirmed or already checked-in affected by flight delays and for passengers affected by overbooking situations. After detecting one of these events it will request to the organization a solution. Traces previous requests and avoids duplicates, until receive a message regarding the status of the request.</p>
<p>Protocols and Activities: <u>CheckForNewPaxEvents</u>, <u>UpdatePaxEventStatus</u>, informsPaxEvent, reportPaxEventStatus</p>
<p>Permissions: reads ResInfoSys // to obtain check-in status. create, read, update Pax Events Queue Data Class // keeps a record of events status</p>
<p>Responsibilities Liveness: PaxMonitor = (<u>CheckForNewPaxEvents</u>^W.informsPaxEvent)^W (reportPaxEventStatus^W.<u>UpdatePaxEventStatus</u>)^W</p> <p>Safety: - successful_connection_with_ResInfoSys = true - successful_connection_with_PaxEvents = true - new_pax_request <> existing_unclosed_pax_request</p>

Table 54 - PaxMonitor (PM) role

Role Schema: PaxFind (PF)
Description: Tries to find the best solution for the events detected and broadcasted by the PaxMonitor role. Suggest a plan to be followed by the company, including checking-in in others flights (even from others companies) and send passengers to the hotel if appropriate. Informs role Operational Control Supervisor of the solution.
Protocols and Activities: <u>FindPaxSolution</u> , informsPaxEvent, sendsPaxSolution, reportPaxEventStatus
Permissions: reads ResInfoSys // to obtain check-in status and flight availability reads AirpInfoSys // to obtain flight information.
Responsibilities Liveness: PaxFind = (informsPaxEvent ^W . <u>FindPaxSolution</u> .[sendsPaxSolution].reportPaxEventStatus) ^W
Safety: - successful_connection_with_ResInfoSys = true - successful_connection_with_AirpInfoSys = true - number_of_pax_requests >= 1 // at least one pax request must exist - request_pax_status = open // processes only open requests - request_pax_status_after_find = close // if unable to find a pax solution - request_pax_status_after_find = found // if a pax solution was found

Table 55 - PaxFind (PF) role

Role Schema: PaxApply (PA)
Description: Applies the pax solution authorized by OperationalControlSupervisor. Before applying the solution checks if the environment is still the same when the problem was detected and/or when the solution was found. If the environment has changed in a way that does not allow applying the solution (or makes the solution unnecessary) informs the OperationalControlSupervisor. After applying successfully the solution informs the organization.
Protocols and Activities: <u>CheckPaxEnvironment</u> , <u>AssignPax</u> , applyPaxSolution, reportPaxSolutionStatus
Permissions: create, update, delete ResInfoSys // to do the check-in for the passengers
Responsibilities Liveness: PaxApply = (applyPaxSolution ^W . <u>CheckPaxEnvironment</u> .[reportPaxSolutionStatus]. <u>AssignPax</u> .reportPaxSolutionStatus) ^W
Safety: - successful_connection_with_ResInfoSys = true - atomic_transactions = true // changes must be atomic - request_pax_status_after_assign = close

Table 56 - PaxApply (PA) role

ANNEX E – INTERACTION MODEL

Protocol name: reportCrewEventStatus		
Initiator (role(s)): CrewFind (CF)	Partner (role(s)): RosterCrewMonitor (RCM)	Input: Close if unable to find a crew member OR Found if a crew member was found; in this case it is waiting for the solution to be applied.
Description: After finishing the task of trying to find a crew member to the open position, it is necessary to inform the status of the request to the Roster Crew Monitor.		Output: OK if able to update the status of the original request OR NOT OK if unable to update the status

Table 57 – Protocol definition for reportCrewEventStatus

Protocol name: sendCrewSolution		
Initiator (role(s)): CrewFind (CF)	Partner (role(s)): OperationsSchedule Manager (OSM)	Input: Crew solution information, namely the list of best crew members that can fill the open position
Description: If a solution (or a list of solutions) is found it is necessary to inform the OperationsScheduleManager that has the control of applying or not the solution, of the best solution according to a utility function.		Output: OK if authorized by the OSM or NOT OK if not authorized.
Extrinsic: The OperationsScheduleManager partner.		

Table 58 – Protocol definition for sendsCrewSolution

Protocol name: requestCrewSolution		
Initiator (role(s)): CrewFind (CF)	Partner (role(s)): TapCRHeuristic (TapCR) OtherCRAAlgorithm (OtherCR)	Input: CFP for filling the open position.
Description: After detecting an open position and defining the characteristics of the necessary crew member to fill that position, starts a CFP to all the specialized roles in finding solutions to this type of problem.		Output: Proposed solution (crew member or a list of crew members) or denied if not able to find a solution.
Extrinsic: The all protocol is extrinsic due to the fact of introducing CrewFind as a controller to coordinate and choose the best solution from the solutions found by the specialized roles.		

Table 59 – Protocol definition for requestsCrewSolution

Protocol name: applyCrewSolution		
Initiator (role(s)): OperationsScheduleManager (OSM)	Partner (role(s)): CrewAssign (CA)	Input: Authorized Crew solution information.
Description: After authorized it is necessary to apply the solution. This protocol initiates the request for applying the solution to the CrewAssign role.		Output: Yes; I will apply the solution (means the environment conditions remains) OR No; I cannot apply the solution (environment changed).
Extrinsic: The OperationsScheduleManager partner.		

Table 60 – Protocol definition for applyCrewSolution

Protocol name: reportCrewSolutionStatus		
Initiator (role(s)): CrewAssign (CA)	Partner (role(s)): OperationsScheduleManager (OSM) RosterCrewMonitor (RCM)	Input: Close request
Description: After applying the solution it is necessary to inform the status of the requests. This protocol enables to share that information with both roles.		Output: OK if able to update the status of the original request OR NOT OK if unable to update the status
Extrinsic: The OperationsScheduleManager partner.		

Table 61 – Protocol definition for reportCrewSolutionStatus

Protocol name: informACEvent		
Initiator (role(s)): RosterAircraftMonitor (RAM)	Partner (role(s)): AircraftFind (AF)	Input: Flight Delay Information
Description: After a flight delay has been detected it is necessary to find a solution to the problem. For that it is necessary to send details about the problem (technical problem, ATC problem, weather problem, etc.) for a solution to be found.		Output: Yes; I will try to find a solution OR No; I cannot process the request (see safety conditions on AircraftFind role).

Table 62 – Protocol definition for informsACEvent

Protocol name: reportACEventStatus		
Initiator (role(s)): AircraftFind (AF)	Partner (role(s)): RosterAircraftMonitor (RAM)	Input: Close if unable to find a solution OR Found if a solution was found; in this case it is waiting for the solution to be applied.
Description: After finishing the task of trying to find a solution to the problem, it is necessary to inform the status of the request to the Aircraft Crew Monitor.		Output: OK if able to update the status of the original request OR NOT OK if unable to update the status

Table 63 - Protocol definition for reportACEventStatus

Protocol name: sendACSolution		
Initiator (role(s)): AircraftFind (AF)	Partner (role(s)): OperationalControl Supervisor (OCS)	Input: Aircraft solution information.
Description: If a solution is found it is necessary to inform the OCS that has the control of applying or not the solution, of the best solution according to a utility function.		Output: OK if authorized by the OCS or NOT OK if not authorized.
Extrinsic: The OperationalControlSupervisor partner		

Table 64 – Protocol definition for sendsACSolution

Protocol name: requestACSolution		
Initiator (role(s)): AircraftFind (AF)	Partner (role(s)): TapARHeuristic (TapAR) OtherARAAlgorithm (OtherAR)	Input: CFP for a solution.
Description: After detecting a flight delay it is necessary to find a solution to the problem. Start a CFP to all specialized roles in finding solutions to this type of problem.		Output: Proposed solution or deny if not able to find a solution.
Extrinsic: The all protocol is extrinsic due to the fact of introducing AircraftFind as a controller to coordinate and choose the best solution from the solutions found by the specialized roles.		

Table 65 – Protocol definition for requestsACSolution

Protocol name: applyACSolution		
Initiator (role(s)): OperationalControlSupervisor (OCS)	Partner (role(s)): AircraftAssign (AA)	Input: Authorized aircraft solution information
Description: After authorized it is necessary to apply the solution. This protocol initiates the request for applying the solution to the AircraftAssign role.		Output: Yes; I will apply the solution (means the environment conditions remains) OR No; I cannot apply the solution (environment changed).
Extrinsic: The OperationalControlSupervisor		

Table 66 - Protocol definition for applyACSolution

Protocol name: reportACSolutionStatus		
Initiator (role(s)): AircraftAssign (AA)	Partner (role(s)): OperationalControlSupervisor (OCS) RosterAircraftMonitor (RAM)	Input: Close request
Description: After applying the solution it is necessary to inform the status of the requests. This protocol enables to share that information with both roles.		Output: OK if able to update the status of the original request OR NOT OK if unable to update the status
Extrinsic: The OperationalControlSupervisor partner.		

Table 67 - Protocol definition for reportACSolutionStatus

Protocol name: informPaxEvent		
Initiator (role(s)): PaxMonitor (PM)	Partner (role(s)): PaxFind (PF)	Input: Pax problem information
Description: After an event has been detected (pax affected by delays and/or overbooking) it is necessary to find a solution. For that it is necessary to send details about the pax problem so that a solution might be found, usually in the form of a suggested plan to be followed by the company.		Output: Yes; I will try to find a solution OR No; I cannot process the request (see safety conditions on CrewFind role).

Table 68 - Protocol definition for informsPaxEvent

Protocol name: reportPaxEventStatus		
Initiator (role(s)): PaxFind (PF)	Partner (role(s)): PaxMonitor (PM)	Input: Close if unable to find a solution OR Found if a plan was found; in this case it is waiting for the solution to be applied.
Description: After finishing the task of trying to a plan to be followed, it is necessary to inform the status of the request to the Pax Monitor.		Output: OK if able to update the status of the original request OR NOT OK if unable to update the status

Table 69 - Protocol definition for reportPaxEventStatus

Protocol name: sendPaxSolution		
Initiator (role(s)): PaxFind (PF)	Partner (role(s)): OperationalControl Supervisor (OCS)	Input: Pax plan solution information.
Description: If a solution is found it is necessary to inform the OCS that has the control of applying or not the solution.		Output: OK if authorized OR NOT OK if not authorized.
Extrinsic: The OperationalControlSupervisor partner.		

Table 70 – Protocol definition for sendsPaxSolution

Protocol name: applyPaxSolution		
Initiator (role(s)): OperationalControlSupervisor (OCS)	Partner (role(s)): PaxApply (PA)	Input: Authorized pax plan information
Description: After authorized it is necessary to apply the solution. This protocol initiates the request for applying the solution to the PaxApply role.		Output: Yes; I will apply the solution (means the environment conditions remains) OR No; I cannot apply the solution (environment changed).
Extrinsic: The OperationalControlSupervisor (OCS) partner.		

Table 71 - Protocol definition for applyPaxSolution

Protocol name: reportPaxSolutionStatus		
Initiator (role(s)): PaxApply (PA)	Partner (role(s)): OperationalControlSupervisor (OCS) PaxMonitor (PM)	Input: Close request
Description: After applying the solution it is necessary to inform the status of the requests. This protocol enables to share that information with both roles.		Output: OK if able to update the status of the original request OR NOT OK if unable to update the status
Extrinsic: The OperationalControlSupervisor partner.		

Table 72 - Protocol definition for reportPaxSolutionStatus

ANNEX F – SERVICE MODEL

Service	Input	Output	Pre-condition	Post-condition
Check crew environment	Authorized list of crew members to be assigned	Status of the environment (Yes if environment conditions remain or NO if environment changed).	Successful connection with Roster and Pairings.	
Assign crew	Authorized list of crew members to be assigned	Final status of the crew event.	Status of the environment = OK.	A successful change of the roster and pairings resource (atomic)
Check aircraft environment	Authorized aircraft solution	Status of the environment	Successful connection with AircRoster.	
Assign aircraft	Authorized aircraft solution	Final status of the aircraft event.	Status of the environment = OK.	A successful change of the AircRoster resource (atomic).
Check pax environment	Authorized pax plan.	Status of the environment	Successful connection with ResInfoSys.	
Assign pax	Authorized pax plan.	Final status of the pax event.	Status of the environment OK.	A successful change of the resInfoSys resource (atomic).

Table 73 – Service Model for agent class OpAssign

Service	Input	Output	Pre-condition	Post-condition
Obtain crew solution authorization	List of crew members to be assigned	Authorization status (OK or NOT OK)	At least one crew solution found	User confirms/does not confirm authorization.
Request crew solution application	Authorized list of crew members to be assigned	Request status (YES = solution can be applied, NO = solution cannot be applied)	Authorization status = OK	User sees status of the request on the screen.
Obtain AC solution authorization	Aircraft solution.	Authorization status	At least one aircraft solution found.	User confirms/does not confirm authorization.
Request AC solution application	Authorized aircraft solution.	Request status	Authorization status = OK.	User sees status of the request on the screen.
Obtain pax solution authorization	Pax solution	Authorization status.	At least one pax solution found	User confirms/does not confirm authorization.
Request pax solution application	Authorized pax solution.	Request status	Authorization status = OK.	User sees status of the request on the screen.

Table 74 – Service Model for agent class OpManager

Service	Input	Output	Pre-condition	Post-condition
Find crew solution	A list of: DutyID, crew number, prng number, list of open positions, eventID.	A list of crew members that can be the solution or DENIED if not found a solution.	Processes only open requests.	
Choose best crew solution	List of crew members that can be assigned.	The best crew members choose according to the utility function.	A solution has been found	Event status changed accordingly.

Table 75 – Service Model for agent class OpCRFind

Service	Input	Output	Pre-condition	Post-condition
Find aircraft solution	A list of: AcReg, FltID, TypeAC, FltStatus	A list of aircraft solutions	Processes only open requests	
Choose best aircraft solution	List of aircraft solutions	The best aircraft solution according to the utility function	A solution has been found	Event status changed accordingly

Table 76 – Service Model for agent class OpARFind

Service	Input	Output	Pre-condition	Post-condition
Find pax solution	A list of: ResNumber, ResStatus, PaxID, PaxName, FltNumber, FltCompany, FltDepAirp, FltArrAirp.	A pax plan solution.	Successful connection with ResInfoSys and AirInfoSys resources; Processes only open requests.	Event status changed accordingly.

Table 77 – Service Model for agent class OpPXFind

Service	Input	Output	Pre-condition	Post-condition
Obtain crew list from heuristic	Prng number, requested duty start date and time, requested duty ready date and time, crew position, other crew related characteristics	A list of crew members that can be a solution, including the associated cost (BaseID, CrewNumber, cost, other related characteristics).	Successful connection with Roster and Pairings resources	Proposed solution or denied if not able to found a solution.

Table 78 – Service Model for agent class OpTapCRH

Service	Input	Output	Pre-condition	Post-condition
Obtain crew list from other algorithm	Prng number, requested duty start date and time, requested duty ready date and time, crew position, other crew related characteristics	A list of crew members that can be a solution, including the associated cost (BaseID, CrewNumber, cost, other related characteristics).	Successful connection with Roster and Pairings resources	Proposed solution or denied if not able to found a solution.

Table 79 – Service Model for agent class OpOtherCRA

Service	Input	Output	Pre-condition	Post-condition
Obtain aircraft list from heuristic	AcReg, FltID, FltStatus, requested airport, requested aircraft family, requested flight date and time, slack time.	A list of aircrafts that can be a solution including costs.	Successful connection with AirRoster resources.	Proposed solution or denied if not able to found a solution.

Table 80 – Service Model for agent OpTapARH

Service	Input	Output	Pre-condition	Post-condition
Obtain aircraft list from other algorithm.	AcReg, FltID, FltStatus, requested airport, requested aircraft family, requested flight date and time, slack time.	A list of aircrafts that can be a solution including costs.	Successful connection with AirRoster resources.	Proposed solution or denied if not able to found a solution.

Table 81 – Service Model for agent class OpOtherARH

INDEX

A

Agents · **23**
 Aircraft Maintenance · **23**
 Aircraft Scheduling · **23**
 airline operation · **23**
 Airline Operations Control Centre · **23**
 Airline Scheduling Problem · **24**

C

consecutive critical periods · **25**
 crew pairing · **25**
 Crew Recovery · **28**
 crew rostering · **25**
 Crew Schedule Manager · **29**
 Crew Scheduling · **23**

D

Disruption Management · **23**
 Distributed Multi-Agent System · **23**
 duty period · **25**

E

effectiveness · **33**
 extra-crew · **30**

F

flight legs · **25**

G

Gaia · **37, 39**
 Gaia2JADE · **39**

H

hour salary · **31**

J

JADE · **38, 39, 45, 81, 84, 107**

M

MaSE · **37**
 Message/UML · **37**
 Multi-Agent Systems · **23**

O

Operations Control · **23**
 Operations Control Supervisor · **29**
 Operations Recovery · **23**
 Operations Recovery Problem · **24**

P

pairings · **25**
 Passenger Recovery · **28**
 Passi · **37**
 per diem value · **31**
 Prometheus · **37**
 published schedule · **31**

R

reserves · **25**
 rest time · **25**
 roster · **32**

S

stand by crew members · **32**
 stand by duties · **25**
 stand by roster · **32**

T

tail numbers · **26**
 Tropos · **37**