The rationale behind the development of an airline operations control centre using Gaia-based methodology

António Castro*

LIACC-NIAD&R
Faculty of Engineering
University of Porto
4200–465, Portugal
E-mail: ajmc@fe.up.pt
and
TAP Portugal
Room 27, Floor 6, Building 27 South
Lisbon, Portugal
E-mail: ancastro@tap.pt
*Corresponding author

Eugénio Oliveira

LIACC-NIAD&R
Faculty of Engineering
University of Porto
4200–465, Portugal
E-mail: eco@fe.up.pt

Abstract: In this paper, we report how we complemented Gaia methodology to analyse and design a multi-agent system for an airline company operations control centre. 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 the representation of Gaia deliverables, such as organisational structure, role and interaction model, and agent and service model.

Keywords: multi-agent systems; agent-oriented methodologies; goal-oriented requirements analysis; operations recovery; disruption management.

Reference to this paper should be made as follows: Castro, A. and Oliveira, E. (2008) 'The rationale behind the development of an airline operations control centre using Gaia-based methodology', *Int. J. Agent-Oriented Software Engineering*, Vol. 2, No. 3, pp.350–377.

Biographical notes: António Castro is a Software Engineer at TAP Portugal, the Portuguese airline company, where he manages software projects related to crew scheduling and operations control. He is currently a PhD student of Informatics Engineering at the Faculty of Engineering, University of Porto. His current research interests include multi-agent systems, agent-oriented software engineering and distributed systems.

Eugénio Oliveira is a Full Professor at the University of Porto, Portugal. He coordinates a research group on distributed artificial intelligence in the LIACC Lab and a doctoral programme in informatics. He obtained his PhD in Knowledge Engineering/Logic Programming from the Universidade Nova in Lisbon (1984). He was Guest Academic at IBM/IEC, Belgium (1984–1985). He was awarded the Gulbenkian Prize in 1983. He has participated in European and nationally funded projects involving intelligent agents. Agent-based frameworks for B2B, multi-agent learning and agent-based teamwork are current topics of interest. He has published a number of papers in journals and proceedings.

1 Introduction

One of the most important concerns in an airline company is operations control. Through operations control mechanisms, the airline company monitors all the flights, checking if they are following the schedule that was previous defined by other areas of the company. Unfortunately, some problems arise during this phase (Kohl and Karisch, 2004). These problems are related to crew members (for example, a crew member who did not report for duty), aircraft (for example, a malfunction or a delay due to bad weather) and passengers. When any of these problems appear, it is necessary to find solutions for them. The Operations Control Centre (OCC) is composed of teams of people specialised in solving the above problems under the supervision of an operations control manager. Each team has a specific goal (for example, to guarantee that each flight has the necessary crew members) that contributes to the common goal of having the airline operation running with as few problems as possible. The process of solving these problems is known as Disruption Management (Kohl *et al*., 2004) or Operations Recovery.

This paper reports how we analysed and designed a Multi-Agent System (MAS) for the TAP Portugal[1] Operations Control Centre, using Gaia (Zambonelli *et al*., 2003) as the main methodology, and how we used parts of other methodologies, such as Tropos (Bresciani *et al*., 2004), and notations, such as UML 2.0,[2] to complement Gaia. Specifically, this paper points out:

- the rationale behind the analysis, design and implementation of our system
- how we used a goal-oriented early requirements analysis to complement Gaia. We also present the advantages we believe had emerged by using this approach to the modelling phase (Section 3).
- how we used the early requirements analysis to subdivide the system into suborganisations (Section 3.2.1) and how we described and represented the preliminary environment model (Section 3.2.2)
- how we created the preliminary role model as well as the interaction model and how we found it useful to have a graphical representation of the preliminary role and preliminary interaction model together with the environment model (Sections 3.2.3 and 3.2.4)

- how we used the previous models to define the organisational structure of our system and how we represented it in UML 2.0, including how we mapped the abstractions to UML metaclasses plus the stereotypes we had created (Section 3.3.1)
- the steps we made to complete the role and the interaction model (Section 3.3.2) and how we represented those two models in UML 2.0, including the mappings we had done (Section 3.3.3)
- how we defined the agent model and the service model (Sections 3.4.1 and 3.4.2) and how we represent those models in UML 2.0, including how we mapped the abstractions to UML metaclasses (Section 3.4.3)
- the steps we made to implement the system using JADE,[3] including implementation of services using JADE behaviours and the ACL performatives used (Section 4).

In Section 5, we present some conclusions about our work.

2 Agent-oriented methodologies and tools

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 decentralised computation. From the commercial point of view we have started to see some real-world agents and multi-agent systems applications; for example, the Distributed Computing with the Digipede Network,[4] which uses agents to make distributed computing a reality, and the Infomobility Services application from the IST IMAGE project (Moraitis *et al.*, 2003b).

Some agent-oriented software development methodologies have been proposed. Some examples are Message/UML (Caire *et al.*, 2001), Tropos (Bresciani *et al.*, 2004), Prometheus (Padgham and Winikoff, 2002), MaSE (Wood and DeLoach, 2001; DeLoach and Wood, 2001), Passi (Cossentino and Potts, 2002) and Gaia (Zambonelli *et al.*, 2003). For more detailed information on the above methodologies and others that we did not mention here, we recommend the reading of Bergenti *et al.* (2004).

Regarding the analysis and design methodologies to be implemented using JADE (Bellifemine *et al.*, 2004), we have seen some attempts to provide roadmaps (*e.g.*, Moraitis *et al.*, 2003a) and tools (*e.g.*, Cossentino *et al.*, 2003; Gómez-Sanz and Pavón, 2003). Recently, Gaia2JADE (Moraitis and Spanoudakis, 2006) has been proposed as a process in order to develop a real-world MAS analysed 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,[5] adding the JADE development phase and proposing a process that covers the full software development cycle. Tropos has been recently updated and now covers the entire software development cycle, including the implementation phase. Tropos includes a tool called 'TAOM4E – Tool for agent-oriented visual modelling for the Eclipse platform',[6] which supports all those phases, including the generation of the JADE code.

To finalise this section we would like to point out that there have been some proposals to integrate the AUML[7] notation into the Gaia process. For example, in Cernuzzi and Zambonelli (2004) and Cernuzzi *et al.* (2004), the authors integrate AUML in Gaia to improve the modelling of open MAS, specifically replacing the protocol model of Gaia with the Agents Interaction Protocol (AIP). Finally, García-Ojeda *et al.* (2005), in their paper about refining Gaia with AUML, propose, among other things, a representation of the organisational structure of the MAS.

3 Analysis and design of the MAS

3.1 A goal-oriented early requirements analysis

The Gaia methodology (Zambonelli *et al.*, 2003) uses as an input a collection of requirements. The methodology does not propose a method to model these requirements. Although we could just list the requirements of the system-to-be, taken from interviews from the stakeholders and users, we think it is important to have a better understanding of those requirements early in the process. For the MAS we have developed for the airline company (Castro and Oliveira, 2005), we choose to adopt a goal-oriented early requirements analysis. The choice was to follow the early requirements analysis of Tropos (Bresciani *et al.*, 2004). In a goal-oriented early requirements analysis the domain stakeholders are modelled as actors depending on one another to achieve their goals. Plans to be performed and resources to be furnished are also modelled here.

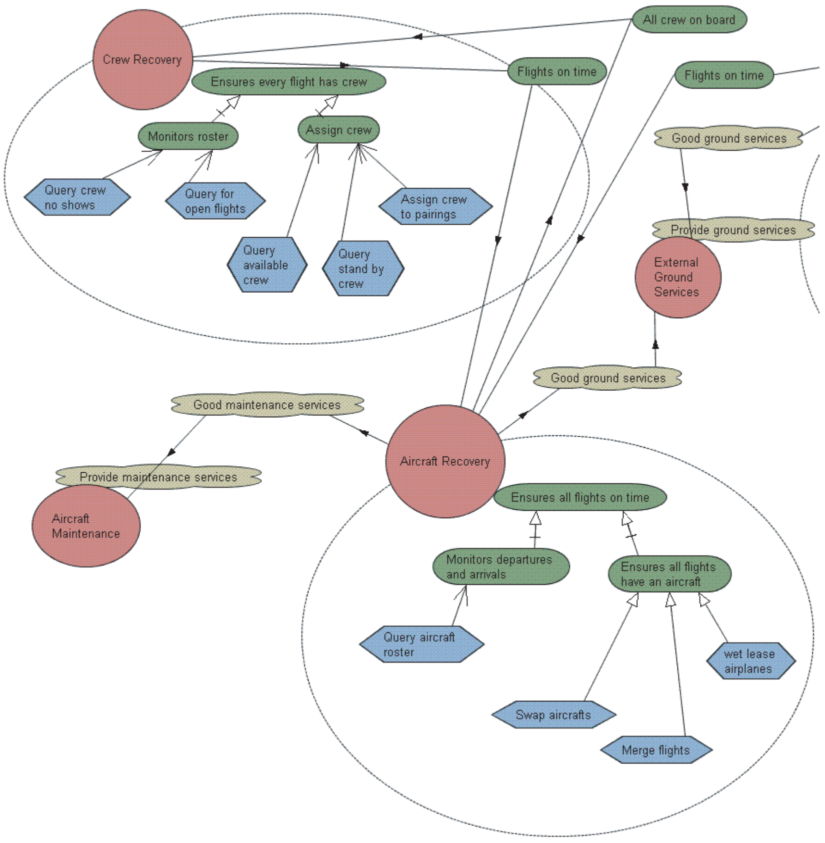
Figure 1 shows a partial Actors and Goals main diagram for an operational base of the airline company, generated during the goal-oriented early requirements analysis and created with the TAOM4E tool. For example, the main goal of the Crew Recovery actor is to 'Ensure every flight has a crew', meaning that before departure, it is necessary to guarantee that all flights have all the crew members assigned according to the regulations. To be able to achieve this goal it is necessary to do an 'and decomposition', meaning that it is necessary to achieve the subgoals 'Monitor Roster' and 'Assign crew'. To fulfil the subgoal '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 who 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 those flights.

To fulfil the subgoal 'Assign crew' it is necessary to execute all these three plans:

1. 'Query available crew' – obtain a list of crew members who 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, he/she can be assigned to the flight.
2. 'Query standby crew' – obtain a list of crew members who have been assigned to a standby activity and who, according to the regulations, can be assigned to the flight.
3. '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/her to the flight.

Figure 1 Actors and Goals main diagram (partial) (see online version for colours)



The execution of the above plans is a means to achieve the previously mentioned goals. To achieve his/her goal the Crew Recovery actor also depends on actor Aircraft Recovery, through the dependency 'Flights on time'.

Figure 1 also shows an example of a soft-goal, that is, a goal without a clear definition and/or criteria for deciding if it is satisfied or not (typically used to model nonfunctional requirements). Actor Aircraft Recovery depends on actor Aircraft Maintenance through the soft-dependency 'Good maintenance services'.

3.1.1 Advantages

We found that the use of a goal-oriented analysis for early requirements helped not only in gathering and understanding the collection of requirements, but also in the analysis phase, as follows:

- The modelling of the specifications, in terms of actors, their roles, their goals and dependencies among them, is more similar to the organisation we have found in the airline OCC. As we stated in the first section, the OCC has the supervision of an operations manager and is composed of teams with specific roles and goals. This correspondence allowed us to better specify the system-to-be.

- In subdividing the system: Modelling the desires, intentions and dependencies of the stakeholders (actors), and specifying the system-to-be in terms of goals and soft-goals, helped in identifying the specific organisations and suborganisations dedicated to the achievement of a given subgoal. The teams in the OCC exhibited behaviours specifically oriented to the achievement of a goal and the corresponding mapping to suborganisations are good examples of this advantage (see Section 3.2.1 for more details).
- In the preliminary role model: identifying the basic skills (functionalities and competences) required by the organisation to achieve its goals. Again, the modelling of the system-to-be in terms of actors (stakeholders) involved and their goals helped to identify the preliminary roles (basic skills). This is in concordance with the statement in Section 4.1.3 in Zambonelli *et al*. (2003).
- In the environment model: having a deeper understanding of the environment where the software must operate as well as of the interactions between software and human agents, during the modelling of the system-to-be, helped in identifying the roles of active components, allowing distinguishing between active components that are only resources from others that should be agentified.
- In the preliminary interaction model: identifying the basic interactions that are required for the exploitation of the basic skills. This is not a direct advantage of applying a goal-oriented early requirements analysis. It occurs from the identification of the basic skills (in the preliminary role model). However, having a better understanding of the actors and their dependencies, as the result of the goal-oriented analysis, during the modelling of the system-to-be facilitates the identification of the basic interactions.

3.2 Analysis

According to Gaia, the analysis phase includes five tasks: (1) subdivision of the system into suborganisations, (2) definition of the environment model, (3) definition of the preliminary role model, (4) definition of the preliminary interaction model and (5) definition of the organisational rules.

3.2.1 Subdividing the system into suborganisations

The suborganisations can be found when there are portions of the overall system that have any of these conditions:

- exhibits a behaviour specifically oriented towards the achievement of a given subgoal
- interacts loosely with other portions of the system
- requires competences that are not needed in other parts of the system.

From the early requirements analysis it is possible to determine three candidate suborganisations that fulfil some of these conditions. The suborganisations identified are described in Table 1.

Table 1 Suborganisations identified

Suborganisation	*Description*
Crew recovery	The subgoal to achieve is 'Ensure that every flight has crew members'. It will loosely interact with other portions of the system because of the dependency 'Flights on time' with the Aircraft Recovery actor.
Aircraft recovery	The subgoal to achieve is 'Ensure all flights are on time'. It will loosely interact with other portions of the system because of the dependencies 'All crew on board' with the Crew Recovery actor, 'Good maintenance services' with the Aircraft Maintenance actor and 'Good ground services' with the External Ground Services actor.
Passenger recovery	The subgoal to achieve is 'Ensure all passengers arrive at the destination'. Interaction depends on 'Flights on time' with the Aircraft Recovery actor and 'Good ground services' with the External Ground Services actor.

3.2.2 Environment model

The first decision to be made is to distinguish between resources and active components. Resources, according to GAIA, 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. Computer-based systems or humans in a process are examples of active components that should not be treated as part of the environment but, instead, should be *agentified.* In our case, we have two humans in the OCC who have an important role in the process (see Table 2) and with which the agents in the MAS will have to interact. These Active Components will be *agentified* by two agents. Table 3 shows some of the resources that are available in the environment.

Table 2 Active components (partial)

Name	*Description*
Operational control supervisor	Final human authority regarding: initiating, cancelling, consolidating or advancing flights, wet lease of airplanes, exchange of airplane or airplane versions, delay of flights, diverting or rerouting flights
Operations and schedule manager	Final human authority regarding: monthly plan management, daily and weekly operation development and crew assignment for unplanned flights and irregularities

Table 3 Resources (partial)

Name	*Description*
Crew sign-on	Contains information regarding the crew sign-on for flights. This will make it possible to know if a crew member did not report for duty. It will allow the implementation of the plan 'query crew no-shows' indicated in the crew recovery goal diagram.
Pairings	Contains information regarding the pairings (and flights) that need to have crew members assigned. It will allow the implementation of the plan 'query for open flights' indicated in the crew recovery goal diagram.
Roster	Contains information regarding the roster of all crew members for a specific operational base. It will allow the implementation of the plans 'query available crew', 'query standby crew' and 'assign crew to pairings' indicated in the crew recovery goal diagram.

For the environment model to be complete, we need to list or represent the resources and actions that will be performed to access them, from the environmental perspective. The tool that we used supports modelling with UML 2.0, including a data diagram, which allows the representation of databases and their relations. An example taken from our environment model is presented in Figure 2. However, in our opinion, this representation is not enough because it does not show in sufficient detail the plans that will be performed using the resources. We have complemented this diagram with the information (partial) shown in Tables 4 and 5.

Figure 2 Environment through a data diagram (partial) (see online version for colours)

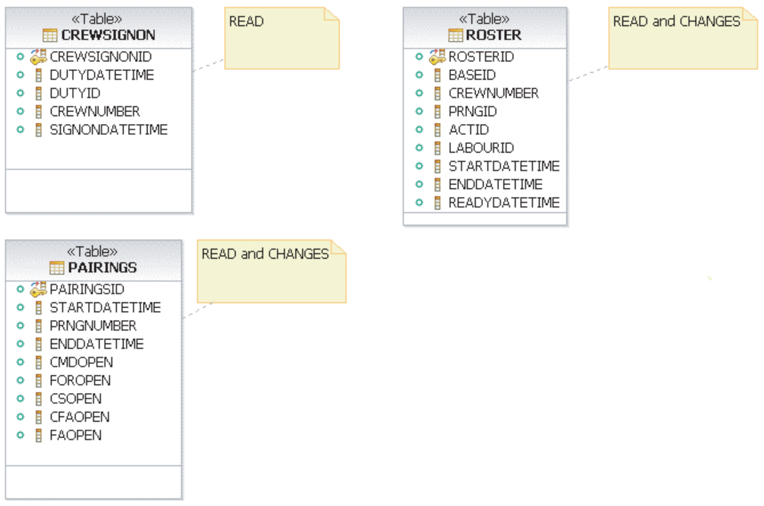


Table 4 CrewSignON resource

Attribute(s)	*Action and Plan(s)*	*Description/Conditions*
DutyID CrewNumber	Action: READ Plan: Query crew no-shows	WHERE (current date and time) >= (DutyDateTime) + min AND (SignOnDateTime) = null

Table 5 Pairings resource

Attribute(s)	*Actions and plan(s)*	*Description/Conditions*
PrngNumber CMDOpen FOROpen CSOpen CFAOpen FAOpen	Action: READ Plan: Query for open flights	WHERE (current date and time) >= (StartDateTime) – hrs AND (cmdopen + foropen + csopen + cfaopen + faopen) > 0
CMDOpen FOROpen CSOpen CFAOpen FAOpen	Action: CHANGES Plan: Assign crew to pairings	Previous value – # crew assigned

Regarding the CrewSignON resource, we can see that the action will be to read information from attributes DutyID and CrewNumber and the plan to be used is 'Query crew no-shows'. That plan executes the read action performing the condition described, that is, it returns the records where the current date is equal or greater than the DutyDateTime plus an additional time (in minutes) and where the SignOnDateTime attribute is null (meaning that the crew did not report for duty). An example of an action that changes the resources is presented in the Pairings Resource table. It is possible to see that the plan 'Assign crew to pairings' will change the attributes CMDOpen, FOROpen, CSOpen, CFAOpen and FAOpen, subtracting from the existing value the number of crew members assigned.

3.2.3 Preliminary role model

In this phase the objective is to identify the basic skills, that is, functionalities and competences, required by the organisation to achieve its goals. Those basic skills are the preliminary roles and we need to identify the ones that will be played whatever the organisational structure that will be adopted later on during the Architectural Design phase. Gaia adopts an abstract, semiformal description to express the capabilities and expected behaviours of the preliminary roles. These are represented by two main classes: Permissions (actions allowed on the environment to accomplish the role) and Responsibilities (attributes that determine the expected behaviour of a role, divided into Liveness Properties and Safety Properties). We recommend reading up on the Gaia methodology (Zambonelli *et al.*, 2003), for more information regarding these classes.

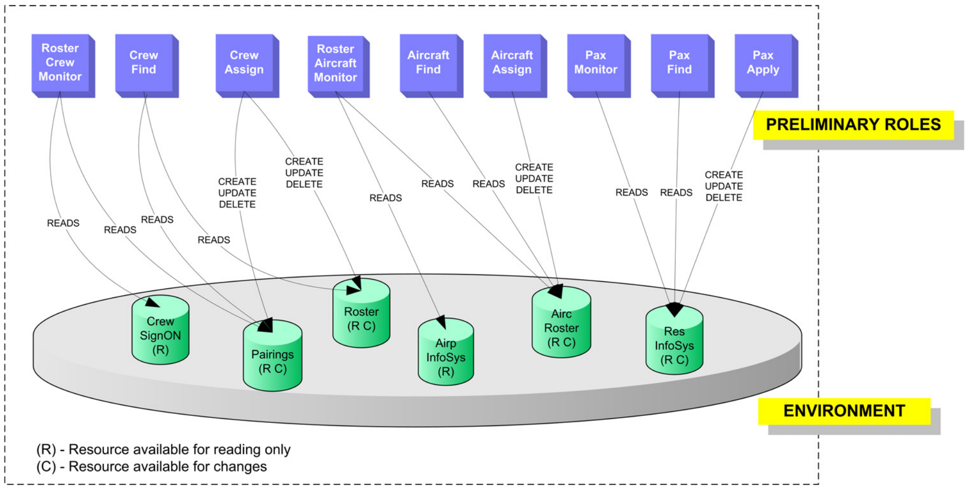
From the 'Actors and goals main diagram' in Figure 1, we can identify several roles that will exist independently of the final organisation of our MAS. A partial list of those roles is:

- *RosterCrewMonitor*, role associated with monitoring the crew roster for events related to crew members who 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.

For each role, it is necessary to define the permissions and the responsibilities and, as an important step, it is necessary to identify any inconsistencies between what operations the environment allows and what the roles (agents) need or must be allowed to do. Gaia refers to the need to create a Role Schema for each role where these properties will be indicated. However, we found that an Environment and Preliminary Roles diagram helps to better identify these inconsistencies.

In the Environment and Preliminary Roles diagram in Figure 3, the operations that the environment allows are labelled with an R for reading and a C for changes, for each resource. Those Actions the roles need or must be allowed to do are indicated by arrows. We can see that the CrewAssign role needs to create/update and/or delete information from the resources Pairings and Roster. Looking to those resources representation, it is possible to see that these operations are allowed. After analysing this diagram, we can see that there are no inconsistencies between the operations allowed by the environment and what the agents need to do.

Figure 3 Environment and preliminary roles diagram (see online version for colours)



To complete the preliminary role model, we have filled a role schema for each of the roles identified with the information collected so far. It is possible to see an example in Table 6.

Table 6 RosterCrewMonitor preliminary role

Role schema: RosterCrewMonitor
Description: This preliminary role involves monitoring the crew roster for events related to crew members not reporting for duty and/or flights with open positions. After detecting one of these events the RosterCrewMonitor will request a solution from the organisation. It should be able to trace previous requests, avoiding duplicates, until it receives a message regarding the status of the request.
Protocols and activities: <u>CheckForNewCrewEvents</u>, <u>UpdateCrewEventStatus</u>
Permissions:
reads CrewSignON // to obtain all who did not report for duty
reads Pairings // to obtain all 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

The preliminary role model will be finished in the Architectural Design, and will be replaced by the full Role Model created during the design.

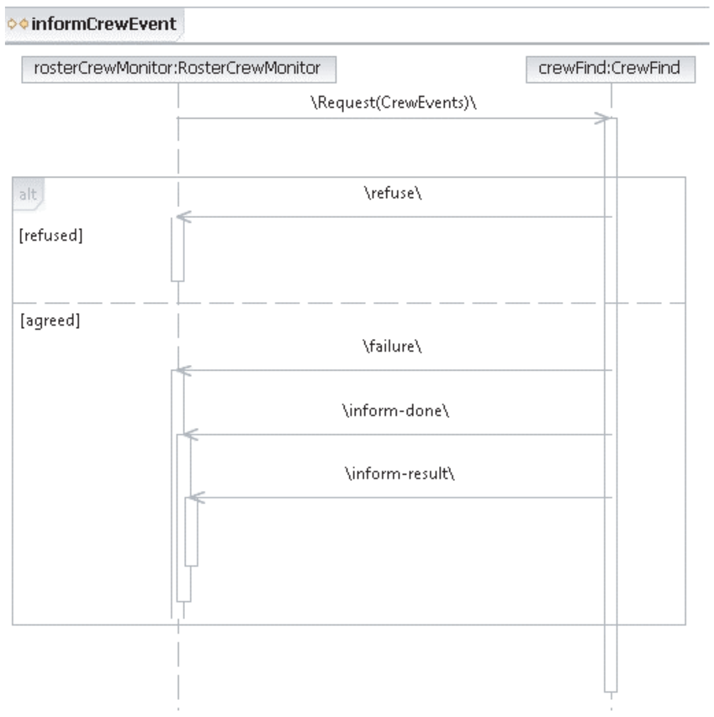
3.2.4 Preliminary interaction model

The objective of this model is to capture the dependencies and relationships between the various roles in the MAS organisation. This is done with one protocol definition for each type of interrole interaction. Table 7 shows the definition of a preliminary protocol using Gaia notation and Figure 4 shows a UML 2.0 interaction diagram for the same protocol.

Table 7 Preliminary protocol informCrewEvents

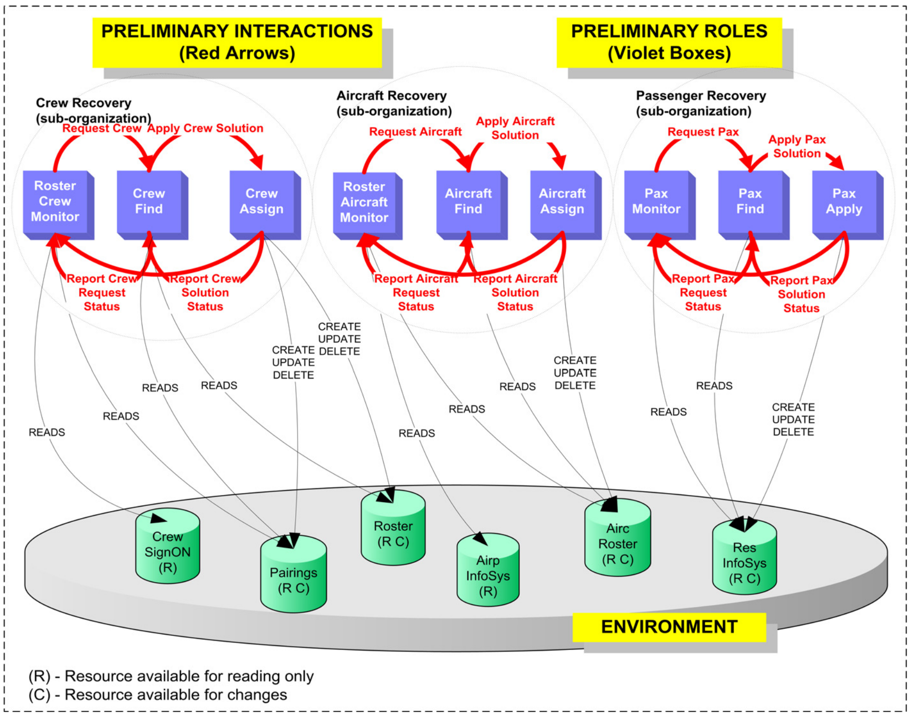
Protocol name: informCrewEvents		
Initiator role(s): RosterCrewMonitor	*Partner role(s):* CrewFind	*Input:* Open position information
Description: After an event has been detected it is necessary to find an available crew member to fill the open position. For that it is necessary to send details about 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).

Figure 4 UML 2.0 interaction diagram for informCrewEvent



To have a better overview of the whole system, we found it useful to complement the Environment and Preliminary Roles diagram with the preliminary interactions (protocols) presented in Figure 5.

Figure 5 Environments, preliminary roles and interactions (see online version for colours)


PRELIMINARY INTERACTIONS (Red Arrows)
PRELIMINARY ROLES (Violet Boxes)
Crew Recovery (sub-organization)
Aircraft Recovery (sub-organization)
Passenger Recovery (sub-organization)
Request Crew
Apply Crew Solution
Request Aircraft
Apply Aircraft Solution
Request Pax
Apply Pax Solution
Roster Crew Monitor
Crew Find
Crew Assign
Roster Aircraft Monitor
Aircraft Find
Aircraft Assign
Pax Monitor
Pax Find
Pax Apply
Report Crew Request Status
Report Crew Solution Status
Report Aircraft Request Status
Report Aircraft Solution Status
Report Pax Request Status
Report Pax Solution Status
READS
CREATE UPDATE DELETE
Crew SignON (R)
Pairings (R C)
Roster (R C)
Airp InfoSys (R)
Airc Roster (R C)
Res InfoSys (R C)
ENVIRONMENT
(R) - Resource available for reading only
(C) - Resource available for changes

3.2.5 *Organisational rules*

According to the authors of Gaia (Zambonelli *et al.*, 2003), "(…) there may be general relationships between roles, between protocols, and between roles and protocols that are best captured by organisational rules". Organisational rules are seen as responsibilities of the organisation as a whole. In the preliminary role model we have already defined or approached the roles' responsibilities. As in that model, organisational rules also have safety and liveness rules, or, as in Zambonelli *et al.* (2001), constraints and relations, respectively:

- *Liveness organisational rules (relations)* define "how the dynamics of the organisation should evolve over time". For example, a specific role can be played by an entity only after it has played a given previous role.
- *Safety organisational rules (constraints)* define "time-independent global invariants for the organisation that must be respected" (Zambonelli *et al.*, 2003). For example, two roles cannot be played by the same entity.

The formalism to express these rules can be the same used for the liveness and safety rules of the roles. They will be expressed by liveness and safety expressions, respectively.

In summary, liveness expressions detail properties related with the dynamics of the organisation, that is, how the execution must evolve; and safety expressions detail properties that must always be true during the whole life of the MAS. A partial list of the liveness organisational rules (relations) we have defined for our system can be found in Table 8, and in Table 9 a partial list of the safety organisational rules (constraints).

Table 8 Liveness rules (relations)

Liveness rules (relations)	*Description*
$applyCrewSolution(CrewAssign(crew(x))) \rightarrow reportCrewSolutionStatus(CrewAssign(crew(x)))$	Protocol *applyCrewSolution* must necessarily be executed by role *CrewAssign* for a specific crew solution before *CrewAssign* can execute protocol *reportCrewSolution* for that crew solution.
$requestCrew(CrewFind(request(x))) \rightarrow reportCrewRequestStatus(CrewFind(request(x)))$	Protocol *requestCrew* must necessarily be executed by role *CrewFind* for a specific request before *CrewFind* can execute protocol *reportCrewRequest* for that request.

Table 9 Safety rules (constraints)

Safety rules (constraints)	*Description*
$\neg(RosterCrewMonitor \mid CrewFind)$	Role *RosterCrewMonitor* and role *CrewFind* can never be played concurrently by the same entity.
$\neg(RosterCrewMonitor \mid CrewAssign)$	Role *RosterCrewMonitor* and role *CrewAssign* can never be played concurrently by the same entity.

3.3 Architectural design

The analysis phase has the objective of understanding what the MAS will have to be. The deliverables of this phase express the functionality and operational environment of the MAS. In the design phase it is necessary to make decisions regarding the actual characteristics of the MAS. So, besides completing and refining the preliminary models, the design will rely, in actual decisions, on the organisational structure and in modelling the MAS based on the specifications produced. The choice of the organisational structure is very important and will affect the development of the succeeding phases. For that we need to choose the desired topology and control regime to be applied. The Gaia paper (Zambonelli *et al*., 2003) has a very good explanation of this important step. We also found it very useful to read the paper from Fox (1981) regarding organisational theory.

3.3.1 Defining the organisational structure

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

- *From the Actors and Goals main diagram* – The main organisation is an operational base with three suborganisations:
  - a Crew recovery
  - b Aircraft recovery
  - c Passenger recovery.

- *From the environment model* – We have identified the following active components (resources that will be agentified), that is, Operational Control Supervisor and Operations and Schedule Manager (both human authorities).
- *From the preliminary role model* – We have identified a requirement that the *CrewFind* role and *AircraftFind* role use different techniques (that is, different algorithms) to find the solutions.
- *From the organisational rules* – We have identified the roles that cannot be played concurrently by the same entity, such as (this is a partial list), *RosterCrewMonitor* and *CrewFind, RosterCrewMonitor* and *CrewAssign, AircraftFind* and *PaxFind.*

With this information, we defined the organisational structure of our MAS. Table 10 gives a summary of the topologies and control regimes applied.

Table 10 Topologies and control regime

Organisation	*Topology*	*Control regime*
Base	Multilevel hierarchy	Mixed: cooperative and authoritative
Crew recovery	Multilevel hierarchy	Work specialisation
Aircraft recovery	Multilevel hierarchy	Work specialisation
Passenger recovery	Hierarchy	Work specialisation

The control regime was defined following the guidelines of Zambonelli *et al.* (2003) and Fox (1981). In the operational base organisation, we have a cooperative control regime between the *Operational Control Supervisor* role and the *Operations Schedule Manager* role due to the *peer relation* between them, and an authoritative control regime from them to the other roles due to the *control relationship* (for example, the Operational Control Supervisor *controls* the Aircraft Assign role). The work specialisation control regime on the other organisations is derived from the fact that the role (for example, *CrewFind* or *AircraftFind*) provides specific services.

To represent the organisational structure, Gaia suggests a coupled adoption of a formal notation and a more intuitive graphical representation. Table 11 is a formal notation of the organisational structure that we defined for the Passenger Recovery suborganisation. Please note that the relationship 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). A combined graphical representation that includes the environment model is presented in Figure 6. A possible representation in UML 2.0 following the suggestions of Bauer and Odell (2005) is presented in Figure 7.

Table 11 Organisational structure for passenger recovery

Statement/Comment
$\forall i, OperationalControlSupervisor \xrightarrow{control} PaxApply[i]$
This means that the role *OperationalControlSupervisor* has an authoritative relationship with role *PaxApply*, controlling, in this case, all the actions of role *PaxApply.* Specifically, role *PaxApply* needs approval from the *OperationalControlSupervisor* before applying the solution. Please note that role *OperationalControlSupervisor* is shared between this suborganisation and the Aircraft Recovery suborganisation.

Table 11 Organisational structure for passenger recovery (continued)

$\forall i, OperationalControlSupervisor \xrightarrow{depends\_on} PaxFind[i]$
This means that the role *OperationalControlSupervisor* relies on resources or knowledge (a solution found to solve a pax recovery problem) from role *PaxFind* to accomplish its task (that is, to authorise or not authorise the assignment of a specific solution).
$\forall i, j, PaxFind[i] \xrightarrow{depends\_on} PaxMonitor[j]$
This means that the role *PaxFind* relies on resources or knowledge (an event related with a pax problem) from the role *PaxMonitor* to accomplish its task (that is, to find a solution to the passenger problem).

Figure 6 Combined graphical representation (see online version for colours)

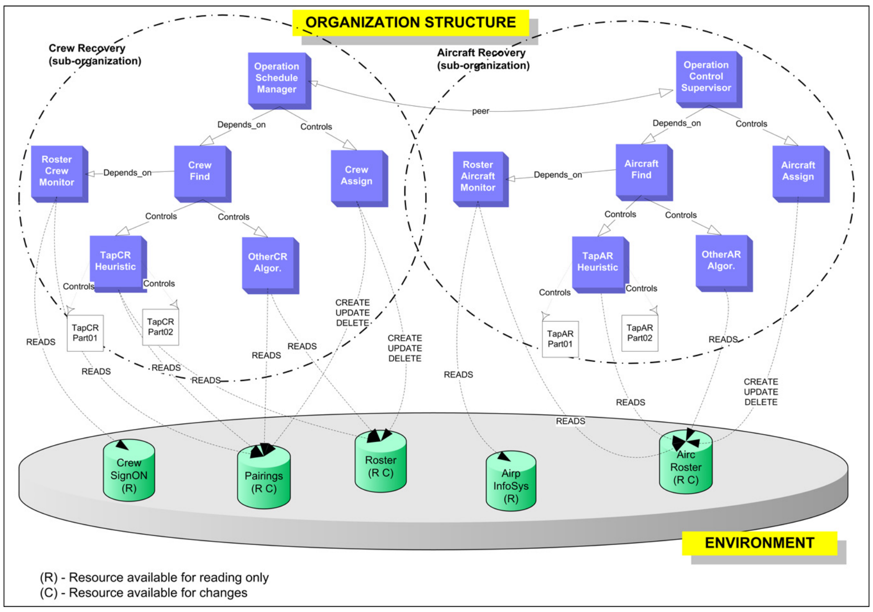
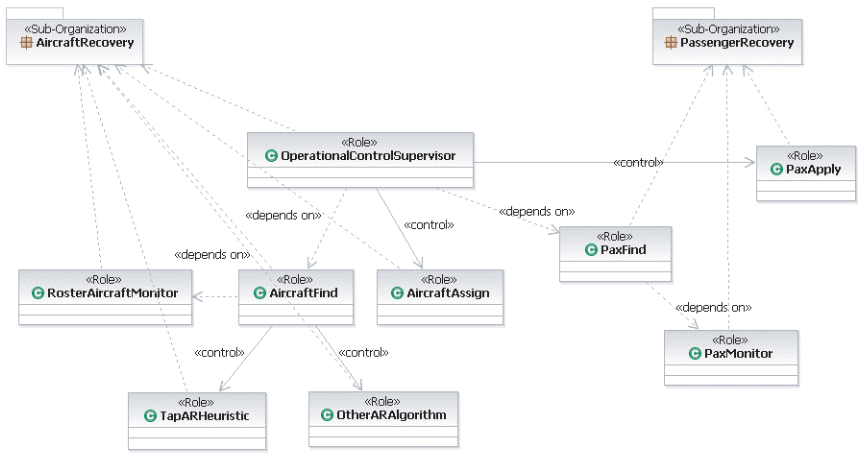


Figure 7 Organisational structure in a UML 2.0 diagram (see online version for colours)



To be able to represent the organisational structure in UML 2.0, we made some mappings between the abstractions used here and the UML 2.0 artefacts as well as created some stereotypes, as follows:

- Organisation abstraction

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

- 'Depends on' abstraction

  We 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 created a *depends on* stereotype associated with the dependency metaclass of UML.

- Controls abstraction

  We mapped this abstraction to an *association* relationship. *Association* relationships in UML are stronger than dependencies and typically indicate that one class retains a relationship with 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 its actions. We created a *control* stereotype associated with the association metaclass of UML.

- Peer abstraction

  We 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 created a *peer* stereotype associated with the dependency metaclass of UML.

- Role abstraction

  We mapped this abstraction to a *class*. A class represents a group of things that have a common state and behaviour. A class can represent a tangible and concrete concept, such as an invoice, or it may be abstract, such as a document. A role represents functionalities and competences that need to be characterised. We found that, at this point, this mapping helped to visualise the roles and the relations among them. However, as stated in Bauer and Odell (2005), "roles cannot be modelled in the necessary detail with any UML 2.0 diagrams".

3.3.2 Completing the role and interaction model

After having the organisational structure, it is possible to complete the role and the interaction model. Some role interactions result from the organisational topology and the protocols that need to be executed from the control regime defined. The tasks that are necessary to be performed to complete both models are:

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

One thing that is important to preserve is the distinction between intrinsic and extrinsic characteristics. Intrinsic characteristics are independent of the use of the role and/or protocol in a specific organisational structure. Extrinsic ones are those that derive from the adoption of a specific organisational structure. This distinction is important in terms of reuse and design for change. Table 12 is an example taken from our complete role model.

Table 12 RosterCrewMonitor role

Role Schema: RosterCrewMonitor

Description: Monitors the crew roster for events related to crew members not reporting for duty and/or flights with open positions. After detecting one of these events, it will request a solution from the organisation. Traces previous requests and avoids duplicates, until it receives a message regarding the status of the request.

Protocols and Activities: <u>CheckForNewCrewEvents</u>, <u>UpdateCrewEventStatus</u>, informsCrewEvent, reportCrewEventStatus

Permissions:

reads CrewSignON // to obtain all who did not report for duty

reads Pairings // to obtain all flights with open positions.

Create, read, update CrewEvents Class

Responsibilities:

Liveness:

RosterCrewMonitor =

(<u>CheckForNewCrewEvents</u>^{W}.informsCrewEvent)^{W} ||

(reportCrewEventStatus^{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

It is important to point out the differences between the final role schema and the preliminary one (from Table 6). First, the liveness responsibilities were completely defined; second, due to the work done during the architectural design, a new resource was identified: CrewEvents. This new resource will keep a record of events status. The permissions property was updated to reflect the access to this new resource. The liveness property specifies what activities and protocols the role will have. They express part of the role's expected behaviour. In our example, *activities* appear underlined and express actions performed by the role that do not involve interaction with any other role (similar to a method in object-oriented terms). *Protocols* are activities that do require interaction with other roles. From the liveness expression of our example:

$$\text{RosterCrewMonitor} = \\ (\underline{\text{CheckForNewCrewEvents}}^{W}.\text{informsCrewEvent})^{W} \,\|\, \\ (\text{reportCrewEventStatus}^{W}.\underline{\text{UpdateCrewEventStatus}})^{W}$$

we can see that role RosterCrewMonitor consists of executing the activity CheckForNewCrewEvents, indefinitely (marked by the ^{W} operator), followed by the execution of the protocol informsCrewEvent. Both of these are performed indefinitely. In parallel (marked by the || operator) it executes the protocol reportCrewEventStatus, indefinitely followed by the activity UpdateCrewEventStatus. Both are also performed indefinitely. To better understand the liveness expressions and the operators used, please consult the GAIA Methodology.

Table 13 shows an example taken from our complete interaction model.

Table 13 sendCrewSolution protocol definition

Protocol name: sendCrewSolution		
Initiator role(s): CrewFind	Partner role(s): OperationsScheduleManager	Input: Crew solution information, namely the list of best crew members who can fill the open positions
Description: If a solution (or a list of solutions) is found it is necessary to inform the OperationsScheduleManager, who has control over applying the solution or not. *Extrinsic:* The OperationsScheduleManager partner		Output: OK if authorised by the Operations Schedule Manager or NOT OK if not authorised.

The important thing to point out here is the distinction that is made regarding the extrinsic characteristic. In this specific example, it is the OperationsScheduleManager partner that is specific to the organisational structure defined. This information might be important if we decide later on to change the organisational structure.

At this stage it is desirable to draw a UML 2.0 interaction diagram similar to the one in Figure 4, for all protocol definitions of our interaction model.

3.3.3 A graphical representation

According to Gaia, the set of role schemas defines the role model and the set of protocols defines the interaction model. A possible representation using UML 2.0 that includes only the Crew Recovery suborganisation is presented in Figure 8. Besides the mappings we did for the organisational structure representation in UML 2.0, in this diagram we did the following:

- Role abstraction

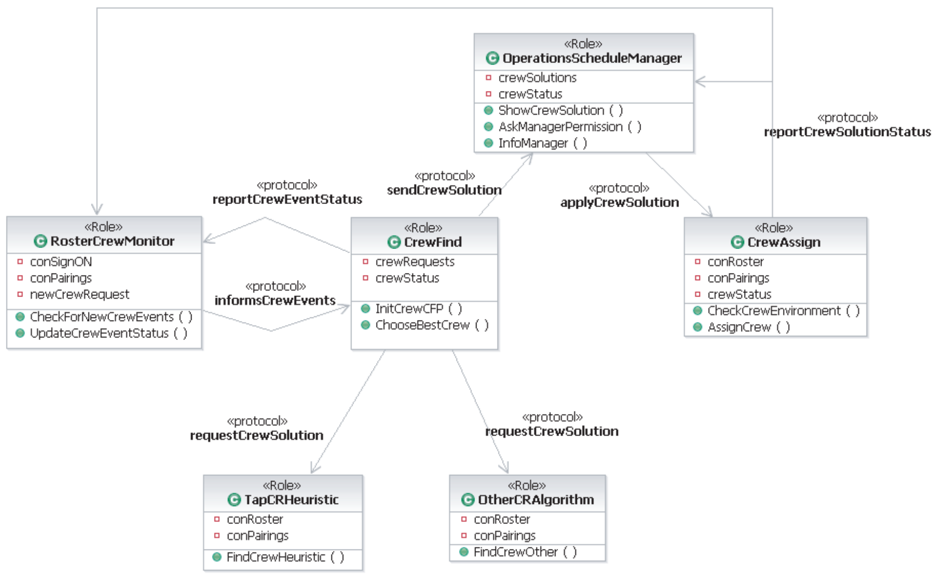
  We complement the previous diagram considering that the usual attributes of the class has part of the safety properties. For example, the role *RosterCrewMonitor* has three attributes: *conSignOn*, *conPairings* and *newCrewRequest*. Those attributes are part of the following safety expressions, respectively: *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 created a *protocol* stereotype associated with 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 visualise the organisation with their roles, activities and protocols, using a commercially available tool.

Figure 8 UML 2.0 role and interaction diagram (see online version for colours)



3.4 Detailed design

In this phase it is necessary to build two final models: Agent model and Service model. These will show the agents that will be implemented as well as the services that will be necessary for each agent to implement. It will be a programming language/middleware-neutral specification.

3.4.1 Agent model

To build the agent model it is possible to 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 make the design compact by reducing the number of classes and instances, leading to a reduction in conceptual complexity. This has to be done without affecting the organisational efficiency, violating the organisational rules and creating 'bounded rationality' problems (that is, without exceeding the amount of information it is possible to process in a given time). GAIA does not specify any special notation for showing the agent model, although it implicitly suggests the adoption of a class model diagram. We have used a simple table, partially presented in Table 14, to specify our agent model.

Table 14 Agent model (partial)

Agent classes/roles
$\textit{OpMonitor}^{1..n}\ \underrightarrow{\textit{play}}\ \textit{RosterCrewMonitor}, \textit{RosterAircraftMonitor}, \textit{PaxMonitor}$
This means that agent class *OpMonitor* will be defined to play the roles *RosterCrewMonitor*, *RosterAircraftMonitor* and *PaxMonitor*, and that we will have between one and *n* instances of this class in our MAS (*n* depends on the number of operational bases defined).
$\textit{OpAssign}^{1..n}\ \underrightarrow{\textit{play}}\ \textit{CrewAssign}, \textit{AircraftAssign}, \textit{PaxApply}$
This means that agent class *OpAssign* will be defined to play the roles *CrewAssign*, *AircraftAssign* and *PaxApply*, and that we will have between one and *n* instances of this class in our MAS (*n* depends on the number of operational bases defined).
$\textit{OpManager}^{1}\ \underrightarrow{\textit{play}}\ \textit{OperationsScheduleManager}, \textit{OperationalControlSupervisor}$
This means that agent class *OpManager* will be defined to play the roles *OperationsScheduleManager*, *AircraftAssign* and *PaxApply*, and we will have one instance of this class in our MAS.

3.4.2 Service model

The services derive from the protocols, activities and liveness expressions of the roles that each agent implements. Usually, there will be one service for each parallel activity of execution that the agent has to execute. According to GAIA, the services model requires that, for each service that may be performed by an agent, four properties are identified: inputs, outputs, preconditions and postconditions. The *inputs* and *outputs* are derived from the interaction model and from the environment model. If the service involves elaboration of data and the exchange of knowledge between the agents, they will come from the protocols. If the service involves evaluation and modification of the environment's resources, they will come from the environment. The *pre- and postconditions* represent restrictions on the execution and completion, respectively, of the services. They derive from the role's safety properties as well as from organisational rules. Applying the above guidelines to our specific problem, we obtain the service model. In Table 15 we present some services for agent class *OpMonitor*, and in Table 16 some services for agent class *OpManager*.

Table 15 Services (partial) of agent class OpMonitor

Service: Monitor Crew Events
Input: Current date, crew slack time, pairing slack time
Output: A list of DutyID, crew number, prng number, list of open positions, eventide
Precondition: Successful connection with CrewSignON and pairing resources
Postcondition: A new crew event that has to be different from an existing unclosed event
Service: Update crew event status
Input: EventID, event status
Output: Number of records updated
Precondition: Successful connection with CrewEvents resource
Postcondition: A successful update of the CrewEvents resource

Table 16 Services (partial) of agent class OpManager

Service: Obtain crew solution authorisation
Input: List of crew members to be assigned
Output: Authorisation status (OK or NOT OK)
Precondition: At least one crew solution found
Postcondition: User confirms or does not confirm authorisation
Service: Request crew solution application
Input: Authorised list of crew members to be assigned
Output: Request status (YES = solution can be applied. NO = solution cannot be applied)
Precondition: Authorisation status = OK
Postcondition: User sees status of the request on the screen

3.4.3 UML 2.0 representation

Figure 9 shows how we represented the agent model in UML 2.0.

- Agent class abstraction

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

  Figure 10 shows how we represent the service model in UML 2.0 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 which 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 operation to be implemented. The agent class *OpMonitor* realises 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 realisation).

- Pre-/Postconditions

  We present the pre-/postconditions using constraints associated with the specific interface. An example for the *MonitorCrewEvents* interface:

  «pre-condition»
  {conn CrewSignON = true; conn Pairings = true}

  «post-condition»
  {new_event <> existing_event}

It is important to note that *Invariant Constraints*, that is, 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, attribute multiplicity and valid values of attributes.

Figure 9 Full agent model in UML 2.0 (see online version for colours)

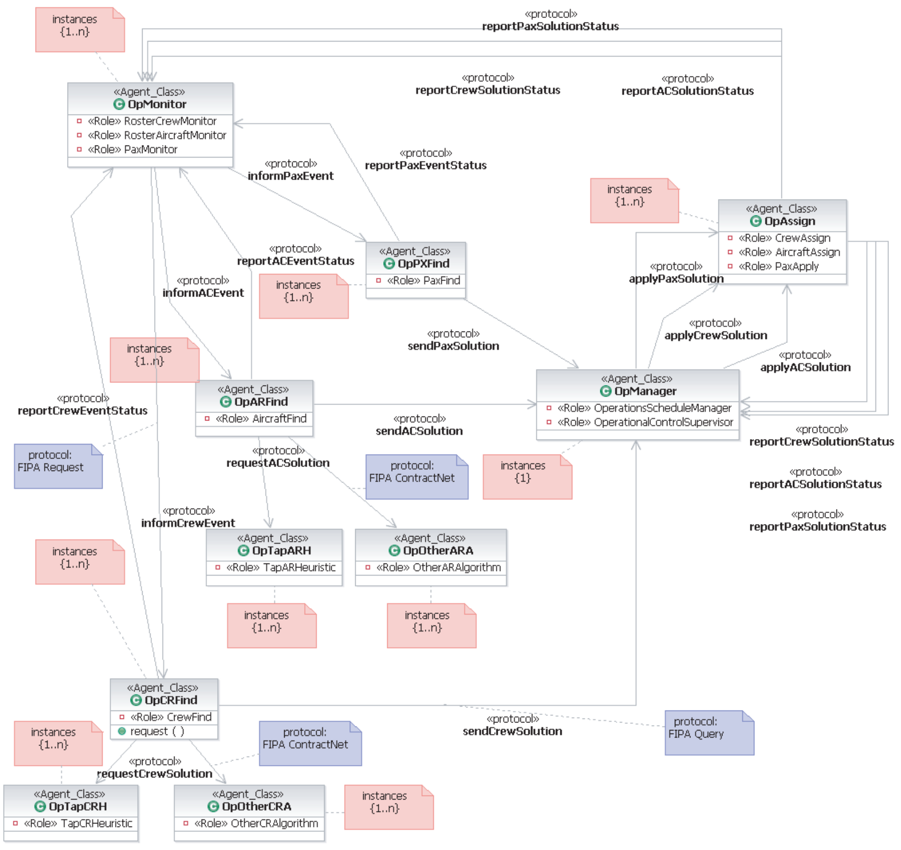
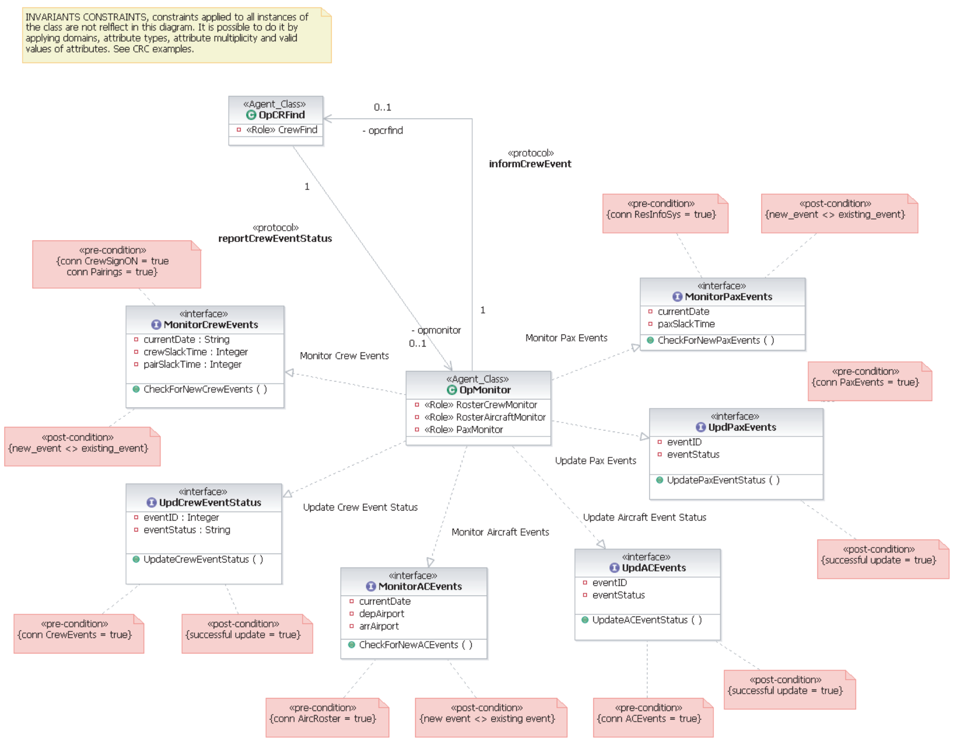

instances {1..n}
«protocol» reportPaxSolutionStatus
«Agent_Class» OpMonitor
«Role» RosterCrewMonitor
«Role» RosterAircraftMonitor
«Role» PaxMonitor
«protocol» reportCrewSolutionStatus
«protocol» reportACSolutionStatus
«protocol» informPaxEvent
«protocol» reportPaxEventStatus
instances {1..n}
«Agent_Class» OpAssign
«Role» CrewAssign
«Role» AircraftAssign
«Role» PaxApply
«protocol» reportACEventStatus
«Agent_Class» OpPXFind
«Role» PaxFind
«protocol» informACEvent
instances {1..n}
«protocol» applyPaxSolution
«protocol» applyCrewSolution
«protocol» sendPaxSolution
«protocol» applyACSolution
instances {1..n}
«Agent_Class» OpARFind
«Role» AircraftFind
«Agent_Class» OpManager
«Role» OperationsScheduleManager
«Role» OperationalControlSupervisor
«protocol» reportCrewEventStatus
«protocol» sendACSolution
protocol: FIPA Request
«protocol» requestACSolution
protocol: FIPA ContractNet
instances {1}
«protocol» reportCrewSolutionStatus
«protocol» reportACSolutionStatus
«protocol» reportPaxSolutionStatus
«protocol» informCrewEvent
instances {1..n}
«Agent_Class» OpTapARH
«Role» TapARHeuristic
«Agent_Class» OpOtherARA
«Role» OtherARAlgorithm
instances {1..n}
instances {1..n}
«Agent_Class» OpCRFind
«Role» CrewFind
request ( )
«protocol» sendCrewSolution
protocol: FIPA Query
instances {1..n}
protocol: FIPA ContractNet
«protocol» requestCrewSolution
«Agent_Class» OpTapCRH
«Role» TapCRHeuristic
«Agent_Class» OpOtherCRA
«Role» OtherCRAlgorithm
instances {1..n}

Figure 10 Service model for OpMonitor (see online version for colours)



4 Implementation

To implement our MAS, we choose the JADE framework.[3] JADE (Bellifemine *et al.*, 2004) is a software development framework written in Java language aimed at the development of MAS. JADE also works as a distributed agent platform across several hosts. Another important feature is that JADE is a FIPA-compliant[8] agent platform and provides implementations of Agent Communication Language (ACL) messages between agents as well as standard interaction protocols (such as FIPA-request and FIPA-query).

To start the implementation and since GAIA produces a technical-neutral specification, it is necessary to map between the detailed design obtained from Gaia and the language/middleware we have used (JADE). We have defined four tasks to be performed:

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, deciding which of the methods (serialised Java objects or extensions of predefined Jade classes) will be the ideal to use.

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 with the possible JADE behaviour 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 also reflect the choice between using an existing JADE protocol and building one.

4.1 Task 1: Concepts and actions

In Task 1, we 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 regarding the content of the messages:

1 the use of strings

2 transmission of serialised Java objects

3 ontology classes taking advantage of the standard FIPA format.

Before deciding which of the methods to use and after reviewing the interaction, environment, agent and service models, we identify the necessary concepts and actions. Some of the concepts and actions are presented in Table 17.

Table 17 Some concepts and actions

Concepts	*Description*
CrewEvent	Characterises a crew event that initiates the process of crew recovery
CrewSolutionList	Characterises 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
CrewSolution	Characterises the crew solution chosen by agent CrewFindAgent and that will be presented to the ManagerAgent for authorisation
Actions	*Description*
ApplyCrewSolution	Action of applying the crew solution after it has been authorised
UpdateEventStatus	Action of making the status update of a crew/aircraft or passenger event

Due to the fact that our agents are developed in Java and that, in this version, our MAS is not an open system and does not need to interoperate with other agent systems, we choose to pass the content of messages as objects.

Although in this real-world project we have mixed the definition of the ontology with the way that ontology will be implemented in the MAS, we believe that it should be done separately: the definition of the ontology should be done during the design phase and the mapping to the technology that will implement it during the implementation phase.

4.2 Task 2: Notation for agents, protocols and services

Regarding Task 2, we present a partial list of the defined notations in Table 18.

Table 18 Agents, protocols and services notations

Design name	*Implementation name*
Agents	
OpMonitor	MonitorAgent
OpCRFind	CrewFindAgent
OpManager	ManagerAgent
Protocols	
informCrewEvent	inform-crew-event
requestCrewSolution	crew-solution-negotiation
applyCrewSolution	request-apply-crew-solution
Services	
Monitor crew events	MonitorCrewEvents
Obtain crew solution authorisation	ObtainCrewAuthorisation
Request crew solution application	RequestApplyCrew

4.3 Task 3: Services and JADE behaviours

All services will be implemented with JADE behaviours that will 'run' inside/or extend a JADE 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 will perform some services indefinitely (for example, monitoring) and/or wait for a message to act (for example, messages that initiate interactions protocols that they need to be part of). Table 19 shows a partial list of the mappings between services and JADE behaviours and FIPA protocols.

Table 19 Mapping (partial) of JADE behaviours and services

Service: MonitorCrewEvents
JADE Behaviour: Ticker
FIPA/JADE IP: fipa-request
Protocol(s) ID: inform-crew-event
Service: FindCrew
JADE Behaviour: Simple
FIPA/JADE IP: fipa-request; fipa-contract-net
Protocol(s) ID: inform-crew-event; crew-solution-negotiation

4.4 Task 4: ACL performatives used

The ACL performatives used are related to the interaction protocols that we have implemented. As an example, the performatives used in the *crew-solution-negotiation* protocol (FIPA contract net) are cfp, refuse, propose, reject_proposal, accept_proposal, failure, inform (done) and inform (result).

5 Conclusion

This paper presents the rationale behind the process we have used to analyse, design and implement a multi-agent system for a real-life, real-size application, an airline company Operations Control Centre, using Gaia (Zambonelli *et al.*, 2003) as the main methodology and JADE (Bellifemine *et al.*, 2004) as the implementation platform.

It is not our goal to present a formal process or a tool for MAS design and implementation. Other works, such as the Gaia2JADE process (Moraitis and Spanoudakis, 2006), Passi (Cossentino and Potts, 2002), Ingenias (Gómez-Sanz and Pavón, 2003) and Tropos (Bresciani *et al.*, 2004), are much more adequate for that.

Additionally, our aim is not to promote the use of Gaia methodology over any other existing methodologies, although we found it easy to understand and apply, and well suited for the analysis and design of MAS.

What we think our work brings to the community of developers who use or want to use Gaia and JADE, is the possibility of seeing how we overcame the difficulties and how we complement some missing parts of the methodology, such as the lack of a process for gathering and modelling requirements.

We also think that the representations we have done using UML 2.0 of some of the deliverables of Gaia (for example, the organisational structure, role and interaction diagrams and agent and service models), including the mappings between the abstractions used in methodologies and the UML concepts, are an added value that our work proposes to the MAS R&D community. However, and to avoid the situation 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
  - a The table notation for the protocol definition (Tables 7 and 13), either in the preliminary or final interaction model, can be replaced by UML 2.0 interaction diagrams (Figure 4) for the same phases.
  - b The formal notation representing the organisational structure (Table 11) can be replaced by the UML 2.0 representation (Figure 7).
  - c The table representation of the agent model (Table 14) can be replaced by a UML 2.0 class diagram as in Figure 9.
  - d The table representation of the service model (Tables 15 and 16) can be replaced by a UML 2.0 class diagram as in Figure 10.
- Used jointly
  - a The combined graphical representation that includes the environmental model of the preliminary role diagram (Figure 3), preliminary interactions diagram (Figure 5) and organisational structure (Figure 6) can be used as a complement to the Gaia preliminary role model, preliminary interaction model and organisational structure UML 2.0 representation (Figure 7), respectively.
  - b The UML 2.0 representation of the role and interaction model (Figure 8) can help to better visualise the organisation with their roles, activities and protocols.

A final added value of this work to the community is the possibility of following the development of a real-world application, from requirements gathering to implementation, and perceiving the rationale that was behind that development.

References

Bauer, B. and Odell, J. (2005) 'UML 2.0 and agents: how to build agent-based systems with the new UML standard', *Journal of Engineering Applications of Artificial Intelligence*, Vol. 18, No. 2, pp.141–157.

Bellifemine, F., Caire, G., Trucco, T. and Rimassa, G. (2004) *JADE Programmer's Guide*, JADE 3.3, TILab S.p.A.

Bergenti, F., Gleizes, M.P. and Zambonelli, F. (2004) *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, Kluwer Academic Press, ISBN: 1402080573.

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J. and Perini, A. (2004) 'Tropos: an agent-oriented software development methodology', *Autonomous Agents and Multi-Agent Systems*, Vol. 8, No. 3, pp.203–236.

Caire, G., Coulier, W., Garijo, F., Gomez, J., Pavon, J., Massonet, P., Leal, F., *et al*. (2001) 'Agent oriented analysis using MESSAGE/UML', *Proceedings of Agent Oriented Software Engineering (AOSE 2001)*, Springer.

Castro, A. and Oliveira, E. (2005) 'A multi-agent system for intelligent monitoring of airline operations', *Proceedings of the Third European Workshop on Multi-Agent Systems*, Brussels, Belgium, pp.91–102.

Cernuzzi, L., Juan, T., Sterling, L. and Zambonelli, F. (2004) 'The GAIA methodology: basic concepts and extensions', *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, Kluwer Academic Press, February, pp.69–88, ISBN: 1402080573.

Cernuzzi, L. and Zambonelli, F. (2004) 'Experiencing AUML in the GAIA methodology', *Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS)*, Porto, Portugal, pp.283–288.

Cossentino, M., Burrafato, P., Lombardo, S. and Sabatucci, L. (2003) 'Introducing pattern reuse in the design of multi-agent systems', in R. Kowalszyk, J.P. Muller, H. Tianfield and R. Unland (Eds.) *Agent Technologies, Infrastructures, Tools, and Applications for E-Services*, LNAI 2592, Berlin: Springer-Verlag, pp.107–120.

Cossentino, M. and Potts, C. (2002) 'A CASE tool supported methodology for the design of multi-agent systems', *Proceedings of the 2002 International Conference on Software Engineering Research and Practice (SERP02)*, Las Vegas, Nevada, USA.

DeLoach, S. and Wood, M. (2001) 'Developing multi-agent systems with agent tool', *Intelligent Agents VII: Agent Theories Architectures and Languages, 7th International Workshop (ATAL 2000)*, LNCS 1986, Boston, MA: Springer-Verlag, pp.46–60.

Fox, M. (1981) 'An organizational view of distributed systems', *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 11, No. 1, pp.70–80.

García-Ojeda, J., Arenas, A. and Pérez-Alcázar, J. (2005) 'Paving the way for implementing multiagent systems: refining GAIA with AUML', *6th International Workshop (AOSE2005)*, LNCS 3950, Berlin: Springer-Verlag, pp.179–189, ISBN: 978-3-540-34097-3.

Gómez-Sanz, J. and Pavón, J. (2003) 'Agent oriented software engineering with INGENIAS', *Multi-agent Systems and Applications III. Proceedings of the 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2003)*, LNCS 2691, Berlin: Springer-Verlag, pp.394–403.

Kohl, N. and Karisch, S. (2004) 'Airline crew rostering: problem types, modelling and optimization', *Annals of Operations Research*, Vol. 127, pp.223–257.

Kohl, N., Larsen, A., Larsen, J., Ross, A. and Tiourline, S. (2004) 'Airline disruption management – perspectives, experiences and outlook', *Carmen Research Technology Report CRTR-0407*, February.

Moraitis, P., Petraki, E. and Spanoudakis, N. (2003a) 'Engineering JADE agents with the GAIA methodology', *Agent Technologies, Infrastructures, Tools, and Applications for E-Services*, R. Kowalszyk, J.P. Muller, H. Tianfield and R. Unland (Eds.) LNAI 2592, Berlin: Springer-Verlag, pp.77–91.

Moraitis, P., Petraki, E. and Spanoudakis, N. (2003b) 'Providing advanced, personalised infomobility services using agent technology', *Proceedings of the 23rd SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence (AI 2003)*, Cambridge, UK, pp.35–48.

Moraitis, P. and Spanoudakis, N. (2006) 'The GAIA2JADE process for multi-agent systems development', *Applied Artificial Intelligence*, Vol. 20, Nos. 2–4, pp.251–273.

Padgham, L. and Winikoff, M. (2002) 'Prometheus: a methodology for developing intelligent agents', *Proceedings of the 3rd International Workshop on Agent Oriented Software Engineering (AOSE 2002)*, Bologna, Italy.

Wood, M. and DeLoach, S. (2001) 'An overview of the multi-agent systems engineering methodology', *Proceedings of the Agent-Oriented Software Engineering First International Workshop (AOSE-2000)*, LNCS 1957, Limerick, Ireland: Springer-Verlag, pp.207–222.

Zambonelli, F., Jennings, N. and Wooldridge, M. (2001) 'Organisational rules as an abstraction for the analysis and design of multi-agent systems', *International Journal of Software Engineering and Knowledge Engineering*, Vol. 11, No. 3, pp.303–328.

Zambonelli, F., Jennings, N. and Wooldridge, M. (2003) 'Developing multi-agent systems: the Gaia methodology', *ACM Transactions on Software Engineering and Methodology*, Vol. 12, No. 3, pp.317–370.

Notes

1 http://www.flytap.com

2 http://www.uml.org

3 http://jade.tilab.com

4 http://www.digipede.net

5 http://www.omg.org

6 http://sra.itc.it/tools/taom4e/

7 http://www.auml.org

8 http://www.fipa.org