Empirical study of coordination strategies in cooperative multi-agent systems

José Arias Fernández
Abstract

Empirical study of coordination strategies in cooperative multi-agent systems

José Arias Fernández

The growing importance of the Internet Of Things in our society leads to the necessity of developing systems where each thing reacts to the environment, taking its own decisions. Agent based systems fit perfectly in resolving those challenges.

The aim of this thesis is to study the communication in multi-agent systems through coordination and negotiation techniques, applying them in a practical case and analyzing the results given after the execution.

The implementation of the study-case has been developed using the Gaia methodology since it is easy, but complex enough to develop a small multi-agent system from an initial set of requirements.

The strategies analyzed are the Global Planning, Structural Organizations, Contractual Networks, Social Election, Coalitions and Negotiation based on Game Theory.

Analysis of the results leads to the conclusion that no strategy is the best in all cases. Each strategy has different strengths, such as efficiency, scalability and robustness. Besides, negotiation techniques yield to poor results when all the agents try to achieve a common goal.
Contents

1 Introduction 4
  1.1 Multi-Agent Systems .................................. 5
  1.1.1 What is an Agent? .................................. 5
  1.1.2 Cooperation and Negotiation ......................... 6
  1.2 The Research Question ................................. 7

2 Background 9
  2.1 Game theory ........................................... 9
  2.1.1 Basic Concepts ..................................... 9
  2.1.2 Strategies .......................................... 12
  2.2 Progressive Boat Rowing with Four Oars System ....... 15
  2.2.1 Race Enrollment .................................... 17
  2.2.2 Rowers .............................................. 17
  2.2.3 Referee .............................................. 18
  2.2.4 Boat Movement ...................................... 19
  2.2.5 Fatigue Control .................................... 21
  2.2.6 Physical State of Rowers ......................... 21

3 Methods 23
  3.1 Methodologies ......................................... 23
  3.1.1 Adelfe .............................................. 23
  3.1.2 Gaia ................................................. 26
  3.1.3 MaSE ............................................... 29
  3.1.4 Prometheus ......................................... 34
  3.1.5 Strengths and Limitations .......................... 36
  3.1.6 Methodology selection - Discussion .............. 38

4 Cooperation and Negotiation Techniques and Strategies 40
  4.1 Coordination ........................................... 40
Chapter 1

Introduction

Nowadays, in a society where the Internet Of Things is gaining more importance over time. Because everything is connected, the necessity of systems where things can work in an autonomous way, observing the environment and reacting to it is increasing. The Multi-Agent programming paradigm is very useful for those systems where there is a changing environment and a large quantity of messages.

This highlights the importance of studying multi-agent systems, and one of the most important aspects of those systems is how their components, the agents, communicate themselves. To study this communication, this thesis proposes a case-study where different techniques are implemented, obtaining results in order to perform an analysis of quality attributes, examining which technique is better depending on the interests of the user.

In this first chapter, an introduction about agents and what is cooperation is given, concluding with a more formal specification of the research question. After this, chapter two presents to the reader information about Game Theory, which helps the reader to understand some definitions used in the thesis and the way in which basic decisions are taken. In this background, the case-study is also presented. With the introduction done, the definition of the methods and techniques that will be used to implement the case-study are presented, giving a brief description of possible multi-agent methodologies in order to select one of them. Chapter four explains the implementation of the system, using the methods and techniques studied before, and obtaining results that will be analyzed in chapter five, which leads to the final chapter, where the conclusions are presented.
1. Introduction

Since the proposed thesis is the study of a multi-agent system, the first step needed is the explanation of what is an agent and how it communicates with others using cooperation and negotiation.

1.1 Multi-Agent Systems

1.1.1 What is an Agent?

Agents, also known as Intelligent Agents, have been defined in many different ways. There is not a standard definition and, in fact, some people reject the word Intelligent for defining them. One possible definition is given by Pitt and Mamdani[1]:

“From the computing perspective, agents are autonomous, asynchronous, communicative distributed and possible mobile processes. From the AI perspective, they are communicative, intelligent, rational and possibly intentional entities.”

But Jennings et al.[2] suggested a different definition:

“An agent is a computer system, situated in some environment, that is capable of flexible autonomous action in this environment in order to meet its design objectives.”

In the first definition it is assumed that agents have to communicate with others (probably agents), but there may be systems with only one agent. In that systems, the agent loses most of the capabilities and, although other approaches would probably be better, it is still being considered an agent. In the first case, we are talking about a Multi-Agent system but it is not necessary to have several agents.

Therefore, the first definition is focused on the capabilities of agents, and the second one is focused on the results.

Considering this situation and in order to meet an agreement, it is important to mention The Foundation for Intelligent Physical Agents (FIPA). It is a IEEE Computer Society standards organization for heterogeneous and interacting agents and agent-based systems that promotes agent-based technology and the interoperability of its standards with other technologies.[3]
1. Introduction

The frameworks used in this thesis, such as JADE[4], use the FIPA[3] standards.

1.1.2 Cooperation and Negotiation

As mentioned above, agents have a communicative aspect. They have to talk with others and this is not a trivial task. These communications have to be done through different cooperation and negotiation techniques.

Coordination is one of the biggest problems in Distributed Artificial Intelligence (DAI). Agents have to think not only about their actions and consequences but also about the actions of other agents. This behaviour requires a communication between all the existing agents in order to act in a coherent way. This is especially crucial when the autonomous agents have different goals. Usually agents use shared resources.[5]

In view of this situation, many techniques have been developed. Most of them can be reformulated in terms of commitments (pledges to undertake a specified course of action) and conventions (means of monitoring commitments in changing circumstances)[6].

Jennings[2] divides the mechanisms in three groups:

1. Organizational Structures that give general, long term information about the relationships between agents.
2. Meta-Level Information Exchange, where agents indicate where they are going to concentrate most of their efforts in order to achieve their goals.
3. Multi-Agent Planning, where agents specify all their future actions before executing them, avoiding inconsistencies.

Durfee[7] considers the coordination as the process where the agent studies other agents’ actions in order to act. Thereby, the actions of the agent depend on the knowledge about the other and, for that, nested models are needed. These are Models about how the other agents see the situation, about how they think others see the situation, about how they think others think how others see the situation and so on. For this reason, in order to anticipate other agents’ actions, the agent needs an untenable amount of information.
Therefore, coordination mechanisms need to delimit part of the knowledge that the agent needs about the others and maintain certain grade of ignorance\cite{7}.

Finally, Huhns and Stephens\cite{8} define the coordination as the property of a system for carrying out an activity in a shared environment. The agents create a model of the future actions, determining the common tasks and sharing knowledge but, what happens when they have opposite goals?.

Huhns and Stephens\cite{8} divided the coordination in two groups: Cooperation and negotiation. The first case is the one assumed in the previous paragraphs where the agents work together in order to achieve goals (common or not), however the negotiation occurs when there are self-interested agents or competitive agents. When they negotiate, they still have to coordinate and communicate between them if they want to obtain results but in this case, the communication, which can be considered as a conversation, will try to maximize the payoff for both agents.

These approaches give the basic information about how the coordination mechanisms work.

### 1.2 The Research Question

The general research question studied in this thesis is: how coordination and negotiation strategies affect the system in terms of:

1. Scalability
2. Efficiency
3. Robustness

The work proposed for this Master Thesis is the comparison from different perspectives, such as robustness, efficiency or scalability, of a variety of strategies for coordinating software agents. With this purpose, after analyzing the different strategies, they will be applied to an implemented scenario in which the selected attributes will be measured and compared. The goal is to obtain empirical evidences of the appropriateness of the collection of strategies selected according to the restriction of the problem and the characteristics of the multi-agent system.
1. Introduction

Scalability is the capability of a system to handle a growing amount of work in a adequate manner or its ability to be enlarged to accommodate that growth[9]. For example, in this particular case, it can refer to the capability of the system to increase the number of agents and the corresponding increase of messages between them.

The concept of scalability is usually difficult to define. For this thesis it will be defined based on the number of agents with which the system can work, considering the number of messages between them.

Efficiency usually describes the completion of a task in terms of effort, cost and time needed. This value, in general, can be measured. Efficiency is different from effectiveness since effectiveness consists of achieving objectives. “Efficiency is doing things right, while Effectiveness is doing the right things”[10]. In the particular case of a race, an efficiency attribute could be the time needed for the completion of the trial.

Robustness is defined as “the ability of a system to resist change without adapting its initial stable configuration”[11]. Basically, it means that the agents have to continue working in the proper way in all the circumstances, even if one of them ends before expected.
Chapter 2

Background

In what follows it is explained in further detail what agents are in order to explain the purpose of this thesis. In this chapter, some game theory concepts are given, with the case-study that will be implemented in order to analyze the coordination and negotiation techniques.

2.1 Game theory

2.1.1 Basic Concepts

Game theory is, basically, a study of strategic decision making. Specifically, it is “the study of mathematical models of conflict and cooperation between intelligent rational decision-makers”[12]. As mentioned above, agents simulate rational decision-makers and, for this reason, Game Theory is of crucial importance for multi-agent systems.

There are two types of games, based on the way agents choose their actions:

1. The normal form:

   (a) Specifies:

   i. The players in game.
   ii. The strategies available to each player. A strategy is an action carried out by the player. A set of strategies are all the possible actions that the player can perform in the game.
   iii. The payoff function that returns the payoff for each player
based on the combination of strategies chosen.

(b) The agents select their actions simultaneously.

2. The extensive form:

(a) Specifies:

i. The players in game.

ii. The set of nodes or states.

iii. A set of functions for each time X that defines: the player who moves at X, the set of possible strategies at X and the previous node or state.

iv. The payoff function that returns the assigned payoff to each player at the node reached.

(b) The agents select their actions consecutively.

An example of the normal form could be the game of Rock-Paper-Scissors. On the other hand, chess is an example of the extensive form.

Because of the scope of this thesis, the normal form will only be considered, as the movement of the agents will be simultaneous (Chapter 2.2).

The games based on the normal form are usually represented as a matrix. In a two-player game, the rows represent the strategies of one agent and the columns the strategies of the other one. The values inside the matrix represent the payoff for each agent for the selected options:

<table>
<thead>
<tr>
<th>Player 2 Option 1</th>
<th>Player 1 Option 1</th>
<th>Player 1 Option 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player 2 Option 1</td>
<td>Player 1 Payoff, Player 2 Payoff</td>
<td>Player 1 Payoff, Player 2 Payoff</td>
</tr>
<tr>
<td>Player 2 Option 2</td>
<td>Player 1 Payoff, Player 2 Payoff</td>
<td>Player 1 Payoff, Player 2 Payoff</td>
</tr>
</tbody>
</table>

Table 2.1: Payoff Matrix

The typical example of a normal form game is the stag hunt game, first suggested by Jean-Jacques Rousseau[13]. In this game, there are two players, called hunters. The hunters have two options:

1. The hunter can shoot the stag.
2. The hunter can shoot the hare.

In this case, the stag is larger than the hare and for that reason the payoff is bigger (with a value of 4), but both hunters have to shoot the stag to catch it. On the other hand, the hunters do not need help if they shoot the hare, but the payoff is smaller (the value is 2).

The payoff matrix would be:

<table>
<thead>
<tr>
<th></th>
<th>Stag</th>
<th>Hare</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stag</td>
<td>4.4</td>
<td>0.2</td>
</tr>
<tr>
<td>Hare</td>
<td>2.0</td>
<td>2.2</td>
</tr>
</tbody>
</table>

Table 2.2: Stag Hunt Payoff Matrix

There are four possible results:

1. If both players shoot the stag so that the stag dies, the payoff is 4 for both of them.

2. If player 1 shoots the stag an player 2 shoots the hare so that the stag survives and the hare dies, the payoff is 0 and 2 for player 1 and 2 respectively.

3. If player 1 shoots the hare an player 2 shoots the stag so that the stag survives and the hare dies, the payoff is 2 and 0 for player 1 and 2 respectively.

4. If both players shoot the stag so that the stag dies, the payoff is 2 for both of them.

Game theory, as it is stated before, studies how the agents will respond in a given game like the stag hunt game.
2. Background

2.1.2 Strategies

Depending on how the agents react, we can define their strategies in several ways.

When the agent have taken the best decision, considering the final result, his behaviour can be defined as best response. In other words, the strategy that maximizes the payoff in a given situation is considered the best response. For instance, in the previous example (stag hunt game), if the first agent shoots the stag, then the best response for the second agent is shooting the stag. On the other hand, if the first agent shoots the hare, then in that case the best response is shooting the hare.

The agent cannot know which one is the best response before acting because the agent does not know what the other agent will do. In addition, when all the agents take the best response, that is called Nash equilibrium[14]. The Nash equilibrium can also be defined when the agent cannot improve the payoff by changing the strategy. In a formal way, Nash Equilibrium can be explained as:

In a game \((S, F)\) with \(n\) players, where \(S_i\) is the players’ \(i\) strategy and \(f_i(x)\) is the payoff function for \(x \in S\). If \(x_i\) is a strategy of player \(i\) and \(x_{-i}\) is a strategy profile of all players except for player \(i\), a strategy profile \(x^* \in S\) is a Nash equilibrium if there are not a different strategy by any single player that makes the change profitable for that player:

\[
\forall i, x_i \in S_i : f_i(x_i^*, x_{-i}^*) \geq f_i(x_i, x_{-i}^*)
\]

The Nash Equilibrium does not mean that the agents have chosen the best action, as it is explained below. The Prisoner’s dilemma Merrill Flood and Melvin Dresher is a very good example to explain the Nash Equilibrium.

The Prisoner’s dilemma consists of a game where two thieves are being interrogated by a policeman. The thieves have two options: to betray the partner, or to deny the participation in the robbery, but they cannot know the decision taken by the other one. When both thieves deny the participation, they go only one year to prison since the policeman does not have proofs. On the other hand, if one of them betrays the other, the betrayed one will go ten years to prison and the betrayer will be free. Finally, if both betray, they will be considered guilty and sent to prison for 5 years.
The following payoff table represents the game:

<table>
<thead>
<tr>
<th></th>
<th>Deny</th>
<th>Betray</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deny</td>
<td>-1,-1</td>
<td>-10,0</td>
</tr>
<tr>
<td>Betray</td>
<td>0,-10</td>
<td>-5,-5</td>
</tr>
</tbody>
</table>

Table 2.3: Prisoner's Dilemma Payoff Matrix

In this case, the Nash Equilibrium corresponds to “betray,betray” because it is the only decision where, if one player changes the decision, the payoff will be worse. If they select “deny,deny” both of them will receive a better payoff, but it is not a Nash Equilibrium because if one of them changes the decision, that player can obtain a better payoff. In addition, if it exist multiple Nash Equilibrium in the game (in the stag hunt game, “stag, stag” and “hare,hare”) we cannot know which equilibrium the agents will choose. Because of these problems, Pareto dominance and optimality concepts were introduced by Vilfredo Pareto[15].

A strategy is strictly Pareto dominated[15] by another strategy when in the second one all the agents receive at least the same payoff than in the first one and one or more agents receive a higher payoff than in the first one. Additionally, a strategy is weakly Pareto dominated by other strategy when all the agents receive at least the same payoff than in the first one. From these definitions, the concept of the Pareto optimality/Pareto efficiency arises. A strategy is Pareto efficient if there are not other strategies that strictly Pareto dominate it. This means that if an agent tries to improve his payoff changing his action, it will reduce the payoff of one or more agents.

In the previous games, the results applying the Pareto strategies are different if applying the Nash equilibrium. In the stag hunt game, there are two Nash equilibriums. It is impossible to know which option the agents will select. However, the strategy “stag-stag” Pareto dominates the other strategy, “hare-hare”, and both agents will benefit if they choose this strategy.

In the Prisoner’s dilemma, the “deny-deny” solution is the Pareto optimal. In fact, all the solutions are Pareto optimal compared to the Nash equilibrium, “betray-betray”.
Although the Pareto optimal solution is better and fairer than in the previous cases, this situation does not happen always. It is guaranteed that the agents will maximize the total payoff, since they are choosing a solution that is not Pareto dominated; however, sometimes, there are situations where the results are not fair. The table 2.4 demonstrates it:

<table>
<thead>
<tr>
<th></th>
<th>Action 1</th>
<th>Action 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action 1</td>
<td>3.2</td>
<td>0.0</td>
</tr>
<tr>
<td>Action 2</td>
<td>0.0</td>
<td>2.3</td>
</tr>
</tbody>
</table>

Table 2.4: Payoff Matrix

In this case, there are two possible options as there are two solutions that are not pareto dominated, the action 1 - action 1 (with a payoff of 3.2) and the action 2 - action 2 (with a payoff of 2.3):

1. Players select one of the not pareto dominated solutions and one of them will receive a lower payoff than the other one, which is not fair.
2. Players select one the pareto dominated solutions, which is useless for both of them.

After this problem, Aumann[16] introduced the correlated equilibrium, where there is an intermediary who recommends the strategy. This intermediary or mediator chooses a strategy randomly, as both agents are in the same situation. When the intermediary recommends a strategy, both agents will follow it, because if the agent that loses payoff select the other one, the payoff will be lower due to miscoordination with the other agent. The problem with this strategy is the necessity of an external entity that decides the results.

The main goal of game theory is to study the rational behaviour in interactive situations (games) between agents (players) in which the results are conditioned by the agent actions. For this reason, the comprehension of these basic game theory terms helps to understand how the agents can take decisions, for instance selecting the Nash Equilibrium or a not pareto dominated solution, in order to reach an agreement. Without this knowledge, agents would try first to satisfy their own necessities without establishing a
2. Background

fair and useful solution. For further information, see [17].

2.2 Progressive Boat Rowing with Four Oars System

The Progressive Boat Rowing with Four Oars System (from now on PB-4) simulates a race between teams composed by four rowers. This approach proposed by Ricardo Imbert as an exercise of the Agent Oriented Software Engineering subject of the European Master on Software Engineering by the Universidad Politecnica de Madrid. This challenge does not correspond to any existing discipline, it is an invented scenario that encourages the cooperation.

The scenario allows different cooperation and negotiation techniques to be implemented in a realistic environment, so different results can be obtained and analyzed, depending on the strategy.

PB-4 boats can be navigated by 1 up to 4 rowers, occupying any of the seats shown in Figure 1, numbered as s1 to s6, taking into account that seats s1, s3, s4 and s6 are intended to be occupied to row with oars o1, o2, o3 and o4, respectively, while seat s2 is used to row with oars o1 and o2 at the same time, and seat s5 to row with oars o3 and o4 at the same time, by a single rower.
PB-4 races have a particular dynamic: the circuit consists of four parts, A, B, C and D, each of them having a dock with the same name at their beginning (see Figure 2.2). Boats start from Dock A with a single rower and, every time the boat arrives to the next dock a new rower boards it, resulting in a team of four rowers in the boat when it crosses the finish line. Rowers on board can occupy any seat during the race, and they are allowed to change or interchange seats at any moment, not necessarily only when boat is at one dock.

As in most competitions, boats start at the same time departing from the START line, and the winner is the boat crossing the FINISH line in the first place. The rest of the positions in the qualification are determined by the order of arrival of the respective boats.

**Figure 2.1:** Schema of a PB-4 boat, seats for rowers accommodation and oars.

**Figure 2.2:** Circuit of a typical PB-4 race.
2.2.1 Race Enrollment

In order to be allowed to participate in a PB-4 race, all the rowers of every team must pass by the registration desk to enroll themselves into the race.

There is a clear time restriction for enrollment before any PB-4 race, and all teams without all four members enrolled when that time expires will be disqualified and not allowed to compete.

Every rower holds a unique number consisting of a combination of his team number (first digit) and a number from 1 to 4 (second digit). For instance, the number for rower “3” of team “9” would be “93”.

During the enrollment procedure, the rower is not only officially showing himself up, but also will communicate his initial position (Dock A, B, C or D).

A team will not be allowed to participate in the race if, by the end of the enrollment period, it has not presented a rower for all the four initial positions. That initial position does not have anything to do with the number of the rower, i.e. rower “93” could be departing from any dock.

By the end of the enrollment period, every rower will receive a notification from the organization indicating: his and his team acceptance for participating in the race; his and his team disqualification from the race, due to the absence of one or more rowers to complete the team or because of formal problems with the registration (not all the docks have a rower assigned).

2.2.2 Rowers

Rowers can only board their boat when it is by the boarding area of the dock in which they are. When boarding the boat, a rower will choose his seat. As it was said before, a rower can change seat as many times as he wants during the race, with the only condition that the destination seat is free.

Boats are initially grabbed to Dock A. They are “released” when the referee gives the signal of start. Of course, any rower could be trying to row before the start signal, but he will not be able to move the boat and he will
be absurdly getting tired.

Once in the boat, a rower can decide:

1. To do nothing.
2. To choose a new free seat (during the time of changing seat, the rower will not be able to do anything else, like, for instance, rowing).
3. To choose an occupied seat, exchanging the seats.
4. To row, in a certain direction (forward or backward), with a certain impulse strength (weak, medium or hard) and, given that some boat seats allow rowing with more than one oar, with a certain oar (o1, o2, o3 or o4).

In addition to these actions, rowers can talk, coordinate themselves, negotiate and make suggestions at any moment.

Finally, every rower has different characteristics in terms of strength, physical resistance and agility, which will determine the evolution of tiredness, impulse and speed of movements during the race.

2.2.3 Referee

The referee is the official member of the organization for the regulation and supervision of the race development. The referee, other than giving the signal of start, is responsible for:

1. Informing the boats when they enter in the boarding area of a dock, allowing the corresponding rowers to board.
2. Informing the boats when they leave the boarding area of a dock, preventing rowers to board them.
3. Deciding boats positions during the race (both partial and final positions).
4. Controlling and informing about timing during the race (both partial and final timing).
5. Informing the boats when they cross the finish line.
2. Background

2.2.4 Boat Movement

Rowing physics are very complex and have an influence on a large number of factors such as the attributes of the rowers, the flow of the river, the wind, etc. To simplify the simulation as much as possible, in PB-4 Agent we will be using the reduced set of ad hoc defined parameters explained below.

As it is shown in Figure 2.3, each time a rower rows an oar, he makes it with a certain impulse that produces two movements: one x, making the boat to turn, and another one y, making the boat to advance. The combination of both forces produces the resultant final movement.

Rowers can decide, at any moment, to row to propel the boat with certain strength. In particular, for the current simulation we will be considering three levels of impulse: weak, medium and strong. The numeric equivalence of these forces will be configurable by the user of the simulation.

However, the final impulse of the boat will be affected by some other factors:
2. Background

Figure 2.3: Strengths affecting the boat movement.

Where:

1. **strength** is the level of strength of the rower (see Section 2.2.5).
2. **tiredness** is the level of tiredness of the rower (see Section 2.2.5).
3. **oar x coef** is a configurable value for adjusting the movement of the boat in the x axis, different for front oars (front oar x coef) and back oars (back oar x coef).
4. **oar y coef** is a configurable value for adjusting the movement of the boat in the y axis, different for front oars (front oar y coef) and back oars (back oar y coef).

The addition of the different forces affecting the boat due to the rowing will result in the final movement of the boat.
2. Background

Rowers should be able to obtain, asking the referee at any moment, information about the boat position, boat speed and direction of movement (according to axis y).

Finally, there will be another configurable attribute, time change seat, determining the total time needed for a rower to change from one seat to another.

2.2.5 Fatigue Control

All data about the state of the rowers is updated each time a rower tries to row. The Fatigue Control calculates the X and Y impulses based on the characteristics of the rower and the values previously stored. All this data is sent to the Boat (if the Rower satisfies the TimeRow and the race have started).

2.2.6 Physical State of Rowers

Rowers will be characterized by three parameters:

1. “strength” with a value between 0 (lower) and 1 (higher), indicating how strong the rower can impulse the boat according to his physical constitution.
2. “resistance,” with a value between 0 and 1, indicating how much the physical effort makes the rower to decrease his performance.
3. “agility,” with a value between 0 and 1, indicating how fast is the rower between one row and the next one.

From these static values, it will be possible to determine the value of the two time-depending elements that will determine the performance of the rowers as time passes.

On the one hand, rowers get more and more tired as they keep on rowing. This is controlled by the function tiredness(t) defined as:

\[
tiredness(0) = 0
\]

\[
tiredness(t) = tiredness(t - 1) + impulse(t) \times (1 - resistance)
\]

being always

\[
tiredness(t) \leq 1.
\]
As time passes, the rower is more tired and, as it was mentioned, his intention of impulsing the boat with a certain strength does not necessarily correspond with the final impulse obtained.

On the other hand, there is a configurable constant minimum number of seconds between rows of a rower, determined by row seconds. However, rowers have some agility, which allow them to row a bit faster or slower. In any case, as time passes, the tiredness(t) has some effect on that speed, ruled by the function timerow(t) as follows:

\[
timerow(0) = \text{rowseconds} - \text{agility} \\
timerow(t) = timerow(t - 1) + \text{tiredness}(t)
\]
Chapter 3

Methods

With the case-study presented, the next step is the implementation. For this implementation, it is necessary to choose a methodology. There are several methodologies that can be applied. Therefore, some of them will be briefly explained and analyzed in this chapter in order to select the more appropriate one to implement the scenario. The coordination and negotiation techniques that will be applied are also explained here.

3.1 Methodologies

There are multiple Multi-Agent Systems methodologies. In this section, there will be a brief introduction of some of the most famous.

3.1.1 Adelfe

Adelfe[18] is based on object oriented methodologies (RUP, Rational Unified Process) using UML[19] and AUML[19] notations and adapting the usual workflows to Multi-Agent Systems. One important point about this methodology is that it is recommended for a specific context, adaptive multi-agent system using the AMAS[20] theory. There are more general methodologies such as GAIA[21] or TROPOS[22].

The methodology has five principal workflows, like most of the object oriented methodologies.
3. Methods

Requirements

First of all, the requirements workflow is needed, a basic step in software engineering. In this step, the system will be defined in order to create a use-case model. Again, it follows the structure of classic methodologies, with functional and non-functional requirements and properties. Basically, the system and environment will be defined. The requirements will be obtained from the interactions between the system and the environment. That is the reason why it is also necessary to model the environment. In order to achieve it, three steps are followed: determination the actors, definition the context and characterization the environment.

Analysis

After the requirements phase, the next one is the analysis workflow. In this case, there are more steps added, such as the identification of the agents and how the agents are adapted to the Adaptive Multi-Agent System theory (AMAS). ADELFE is focused on the identification of the agents. For this objective, ADELFE is highly restrictive and it only accepts specific agents. These agents have to be cooperative and to concord with the AMAS theory. That means that, due to this theory, the global function is not developed within the agents, but also the agents are self-organizative and the global function comes from the interactions and collaborations of the agents. This means that if an aspect of the agents is changed, the global function of the whole system will change without the needing to modify the high level parts.

The agents are composed by five elements:

2. Representations of the World: How they understand the world.
3. Interaction Language: The language used to communicate with others (agents and environment).
4. Aptitudes: The capacities they possess to reason regarding their knowledge.
5. Social Attitude: Communication mechanisms.

In order to define those agents, there are some guidelines.
3. Methods

1. It has to be an entity defined in the requirements phase.

2. The entity has to work with unexpected events and can have different representations about the world (included about itself).

In this way, considering separately each use-case through scenarios, the system is divided in entities. With these acts, a preliminary set of entities is obtained. The next step is to see which of them will be useful in the system. Once this step has been achieved, the methodology recommends to do an extra check to see if this case, this system, is within the limits of the AMAS theory, because not always the multi-agent systems are the right choice. For this purpose, there is an associated tool to ADELFE that provides that information, analyzing answers given by the designer to different questions, in order to tell him/her if using the AMAS theory is useful or not to implement the target system, such as the question “The solution is usually obtained by repeating tests”.

The next step is the definition of agent from the entities. At this point, there have been identified all the entities but not all of them will be agents. It is here where the selection is done, based on the cooperative aspect mentioned above and other agent’s characteristics such as autonomy, locality, capacity of negotiation. The last needed thing is the study of the interaction between the entities.

Design

Finally, in the design phase, the agent models and the cooperation failures (known as ’Non Cooperative Situations”, NCS) are added. From these models, the designer must describe an architecture of the agent, considering his behaviour, but, the system with all the properties and capacities is not defined. According to the AMAS theory, the agents are the elements created, and through this cooperation and interaction, the global function will be determined. When the agent detects failures in his cooperation, it will change his behaviour in order to suit to that situation. In this scope, ADELFE provides help to the designers with generic cooperative failures such as incomprehension, ambiguity, uselessness or conflict.

Regarding the implementation and testing, it is similar to what is done in RUP but additionally, with the decision tool, ADELFE provides a prototyping tool based on the OpenTool software[23].
3.1.2 Gaia

One of the principal differences of Gaia[21] compared to other multi-agent methodologies is its generality, making it applicable to many different cases.

In the first place, a specific requirements phase is not needed. This methodology is applicable from a list of requirements and therefore, it does not matter the way the requirements were obtained. It is an independent phase. Furthermore, the purpose of this methodology is to move from the requirements to a design sufficiently detailed to be implemented directly, by specifying more and more the concepts in each phase.

For this reason, Gaia defines two types of basic concepts: abstract concepts (analysis) and concrete concepts (design):

<table>
<thead>
<tr>
<th>Abstract Concepts</th>
<th>Concrete Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roles</td>
<td>Agent Types</td>
</tr>
<tr>
<td>Permissions</td>
<td>Services</td>
</tr>
<tr>
<td>Responsibilities</td>
<td>Acquaintances</td>
</tr>
<tr>
<td>Protocols</td>
<td></td>
</tr>
<tr>
<td>Activities</td>
<td></td>
</tr>
<tr>
<td>Liveness properties</td>
<td></td>
</tr>
<tr>
<td>Safety properties</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.2: Concepts[18]
3. Methods

Analysis

The Analysis step tries to understand the system and its structure without the need of knowing the details of the implementation. The first concept that emerges is the concept of “system”, the most abstract of all of them. In Gaia it is also known as “society” or “organization” due to the fact that the system is considered as an artificial society or organization. This is because a set of roles which fits with a society point of view must be defined.

In order to define the roles, there are some concepts, which are the main base of the methodology.

1. Responsibilities: Basically, defines the function of the role. There are two types of responsibilities: Liveness properties, that are based on the “good” things that have to happen; and safety properties, that are invariants, and means that nothing bad happens.
2. Permissions: In order to carry out the responsibilities, the role has to have access to certain elements. These accesses are defined in the permissions. In general, there are referred to information access.
3. Activities: The facts performed by the agents in which there is any type of interaction with other agents. Private actions.
4. Protocols: Defines the interactions with other agents.

From this perspective, there are two models. The Roles Model that defines the responsibilities and permissions of the roles, and The Interaction Model that consists of a set of protocols.

Design

In this phase the abstract models obtained in the analysis will be converted to low level models that will allow to implement a solution. The difference with other approaches, like object oriented methodologies, is that it is set according how the society cooperates in order to achieve the system goals and what they need to do it. How agents do it is not important for Gaia. In this process three models will be generated.

The Agent Model

This model will define the different agent types and the agent instances that will be in the system at run-time. These agent types are formed by a set
of agent roles. It could be that each agent type contains only one role, but sometimes, for technical purposes for instance, the same agent type can play several roles. This is a designer choice; the designer has to choose between coherence and efficiency.

Additionally, the number of instances of each agent type in the agent model will be annotated.

The Services Model

In this model, all the services and properties belonging to each role will be explained. An activity of an agent could be an example of service. For each service it is important to identify the inputs, outputs, pre-conditions, and postconditions.

The Acquaintance Model

This model defines the communication links that exist between agent types. The model do not inform about what are the agents sending, just the existence of a path. With this model it is easy to identify possible bottlenecks and it helps to create a system loosely coupled. The model can be represented as a graph.

Following those models, it is easy to implement a solution.
3. Methods

3.1.3 MaSE

The MaSE[19] methodology is a specialization of other traditional software engineering methodologies that has two basic phases: Analysis and design.

Figure 3.3: MaSE Workflow[19]

Despite the figure, the methodology is iterative, allowing the user to move from one step to other without consequences in order to achieve the solution, a consistent system. This represents one of the strong points of MaSe, the ability to track changes throughout the process, focusing on the traceability.
3. Methods

Analysis

Like Gaia, the main goal is the creation of roles that describe what the system is going to do. These roles will carry out functions within the system. Each role will be in charge of achieving or helping to achieve goals and sub-goals. Those goals are a set of requirements. To ensure that the roles fulfill the goals, MaSE recommends use cases.

Therefore, the first phase will mean find those goals that are needed to achieve from an initial specification. Through these requirements, the analyst can know how the system should react according to the different inputs, but here MaSE encourage thinking from the system point of view, instead of the user point of view. The overall goal of the system is to satisfy the desires of the user. If the user wants to know the number of red cars in the road, the system goal would be "to inform the user about the number of red cars in the road". This is done in this way, based on goals, because with a changing environment, where time is especially important, this way is more stable than other information structures. In addition, goals are constant in the whole process of analysis and design, as goals are a very abstract concept.

The recommended way to obtain the goals is through scenarios got from the specification. In this manner, the critical aspects are identified and translated into goals as much abstract as possible. From now, all the roles and task defined must refer to one of the found goals. If that is not the case, then, there are superfluous tasks or it is needed to add a new goal.

The next step is the structuring of the goals, making a Goal Hierarchy Diagram. This diagram is not a tree, as sub-goals can have more than one parent goal. All the goals in the same level of the diagram has to have the same detail level, allowing to manage them easily.

In the first place, we need to place the general goal at the top of the diagram. If this is not possible as there is not a general goal, a new general goal is created, involving all the roles with high abstraction level. From this point, the subgoals are allocated, selecting the goals whose achievement involves the attainment of the parent goal. This does not mean that the goals are divided in steps, for instance, in the previous example, where the main goal was to inform the user about the number of red cars, the activities could be detecting each car, counting the red cars from the list of cars and informing the user of that counter. In contrast, with goals, it does not matter how the
activity is done. One goal could be to detect the red cars (how it is not important) and the other one is to inform. Through this process, if the analyst has to decide how the goal is done, it means that the limit has been reached.

There are four special types of goals:

1. **Summary** - That goal derived from other goals to create a common parent goal. Usually occurs in the high level goals.
2. **Partitioned** - It is a goal with a set of sub-goals that when are done, the goal is achieved. All summary goals are partitioned and annotated differently in the diagram .
3. **Combined** - When goals have similar characteristics and are very close, the goals are grouped in one.
4. **Non-functional** - Goals that do not feet directly with the overall system goal but it is still needed. Usually, for non-functional requirements. A new "branch" is created to refer to those goals.

Goals could have more than one type, but also, goals can have no type.

After that, the next step of the analysis phase is the application of the use-cases. If the use-cases did not exist previously, new use-cases are created. If the use-cases modify the goals, the previous is redone, changing the diagram. The objective of the use-cases is to identify the communication paths for each one of the goals, and to check if there are errors on them.

From this use-cases, sequence diagrams can be obtained where the participants are shown. Each participant will be a role, obtaining the first set of roles for the refining roles step.
In this manner, the roles are obtained from the requirements, which are convenient when designing multi-agent systems. Generally, a one-to-one relationship is established, one role for each goal, but not always. In addition, some goals such as partitioned goals cannot belong to any role, as other roles will carry out goals that will satisfy the partitioned goal. Again, if new goals are discovered, they will be added to the list of goals and the previous steps have to be remade.

All these roles are specified in the MaSE Role Model, including their tasks and interactions with other roles. Roles are identified as rectangles and tasks as circles attached to the role. Between the tasks, there are lines that define the communications. The initiator and responder are identified by arrows at the end of the lines.

Tasks are obtained from the goals and Task Diagrams. These tasks are carrying out by roles in order to achieve goals. There are two types of tasks, persistent tasks and transient tasks. Persistent tasks are those that do not have an event that starts them, generally these tasks are initiated when the agent is created and finish when the agent dies. On the other hand, transient tasks have a trigger that initiates the task. Transient tasks allow to have several tasks executing at the same time. All messages sent through roles and the events emerged from tasks are queued to ensure that all messages are received. For this reason, transitions between states are instantaneous.
To deal with these concurrent tasks, Concurrent Tasks Diagrams are created for each task. Each task defines a part of the global behaviour that the role need follow to accomplish the goal.

An example of user notification is the following:

![Concurrent User Notification Diagram](image)

**Figure 3.6:** Concurrent User Notification Diagram

After the creation of the initial diagram, internal activities are determined in order to explain better each task, giving an extra understanding. With this detailed diagram, all the tasks are studied and merged if it is possible.

**Design**

The Design is done by four steps. Creating the Agent Classes, the conversations between them, Assembling Agent Classes and finally, the definition of the location and number of agents deployed.

In the Creating Agent Classes phase, agent classes are created from the roles previously defined. With this step, a class diagram is obtained, which gives a general information of the agent organization and their basic communications. These classes follow the same style than classes in object orientation. The agents are instances of these classes. In this step only the roles and tasks are defined, not the internal details. Again, it is important to assure that each role has at least one agent class. This allows the designer to do an
3. Methods

3.1.4 Prometheus

This methodology[25] has been developed over the last several years in collaboration with Agent Oriented Software (AOS).

The main goal of this methodology is the possibility to be used by non-expert users. The methodology is based on three phases: The system specification phase, the architectural design phase and the detailed design phase.

In order to specify the system, some concepts need to be explained.

First of all, the information needed and produced (inputs and outputs) come up in order to define the functionalities of the system. The function contains a name, a short natural language description, a list of actions (mech-
anisms that affect the environment), a list of relevant percepts (information that comes from the environment), data used and produced and a brief description of interactions with other functionalities.

It is important to distinguish between percepts and events. Percepts are external information, from the environment. Events are happenings within the system.

These functions are based on particular aspects of the system, extending the information with scenarios that give an extra vision of the system, as a sequence of steps describing an example. Each step has the name of the responsible function and the information produced, so it is possible to assure the consistency.

With these concepts defined, it is possible to have an initial image of the system in order to start with the Architectural design.

The most important decision in this step is to decide which agents will exist. Each function is assigned to a possible agent, identifying agents by grouping functionalities. For this purpose, there are specific guides, for instance, if different functionalities use the same information, they will probably belong to the same agent. In general, all the functionalities are grouped unless a different hardware is used for each one.

After this process, the possible agents are listed and defined like functionalities. The agent descriptor has a name and a description of the agent, a brief description about the ways to interact with other agents, the list of functionalities associated, the possible cardinality, and the data that read and produce.

Again, we have to ensure the consistency with the functionalities. The next step is the identification of the events that will be generated through the percepts, and which agents will react to those events.

All this data can be put together in a system overview diagram with the agents, functions, information and communications path together, which allows to obtain a general understanding of the system as a whole.

Finally, the interactions between agents can be developed with interaction diagrams, similar to sequence diagrams where the objects are agents.
Those diagrams can be translated to protocols diagrams, that explain steps of the interaction diagrams deeper, showing all the possible variations (if the step can fail, the path for the success and for the failure).

The final phase is the detailed design, where the internal structure of the agents is developed.

In order to do that, capabilities, internal events, plans and detailed data structures are defined. The already found functionalities give an initial set of capabilities. Some capabilities can be nested to other capabilities. Each capability has to have a name, a description, which events are inputs and outputs, the interactions with other capabilities or the inclusion of new capabilities, and the information read and produced.

After that, individual plans, events and data descriptors are created. With these last definitions, all the information needed for the implementation is done.

Plans have an identifier, the triggering event type, a description of the steps, the use context and a list of read and written data. Events have a description of their purpose and the information they carry. Data descriptors have the fields and methods of the classes that contains that data.

Finally, the implementation can start.

### 3.1.5 Strengths and Limitations

With all the methodologies explained, in the next paragraphs there is a brief explanation of the strengths and limitations of each technique.

**Adelte strengths**

One of the best characteristics of Adelte is the use of the classical life cycle and use of UML notations. These concepts are known by most of programmers, facilitating the learning and they offer a tool integrated with Open-Tool, that facilitates the work to experienced programmers.
3. Methods

Adelle Limitations
It is not as general as other methodologies and the environment needs to be modeled. In addition, only very specific agents can be developed, following the AMAS theory and the tools offered have very complex graphical interfaces, so that an initial learning period is assumed.

MaSE strengths
All the steps are very well documented and specified. Following those steps, all the system is perfectly defined, from very abstract concepts to concrete concepts.

MaSE Limitations
The main strength is also a limitation. All the steps needed make the methodology stricter than others. The designer has freedom in each step, but there is a unique path to build the system, redoing previous steps continuously. This process is very useful for very big and complex systems, but takes too much time for medium or small systems. Still in those small systems MaSE is a good option, but if the information of the system is available, other methodologies can be developed easily.

Gaia strengths
One big point for Gaia is that the agents can be heterogeneous and may be implemented in different languages and architectures. The designer has a lot of freedom developing the models.

Gaia Limitations
It is assumed that the goal is to obtain a system that maximizes some global quality measure and is not intended for systems that admit the possibility of true conflict, all the agents and roles try to achieve one specific goal. Also, less than 100 agents are expected in the system.
3. Methods

Prometheus strengths

Non-expert users can use it easily.

Prometheus Limitations

In order to satisfy the requirement of an easy methodology for non-expert users, the methodology looks too abstract to develop a system with guarantees of consistency and efficiency.

3.1.6 Methodology selection - Discussion

First of all, it is important to point out that all the methodologies presented could be used to develop the proposed scenario.

As mentioned before, some methodologies (such as MaSE) are quite strict and require following a number of steps to get the solution. On the other hand, other methodologies presented, such as Prometheus, seem to be too abstract to be confident enough in the definition of the system. Therefore, this last option is usually recommended to people with poor technical knowledge, since the methodology has a level low enough to be able to model systems without a broad technical knowledge.

Analyzing the different options, the conclusion that MaSE is a methodology that, iteratively, ensures the possibility of getting to a right solution comes up and, for this reason, it requires revising the models again and again, that is, each time that the next step is reached. Having used this methodology in this project would have required too much time, reducing the time spent in the strategies analysis and in the obtainment of results, which is of major importance. For all these reasons, this methodology was discarded.

Considering this, the opposite case, Prometheus, could have been chosen. However, as technical users, this methodology can be considered too ambiguous to have the certainty of being developing a good model.

All in all, Adelfe and Gaia were the remaining methodologies to choose in between. Gaia was selected as it is a not too complex methodology which major disadvantage- can only be used for less than hundred agents- does not interfere with this project that it is a problem relatively small. Additionally, Gaia offers a complexity enough to do a quite exhaustive definition without
needing too much time to be developed. Based on the scenario, it is possible to appreciate the different possible roles in the system without too much effort required, and, therefore, it is possible to create the models indicated by Gaia, defining clearly the system without a lot of time needed.
Chapter 4

Cooperation and Negotiation
Techniques and Strategies

The strategies selected to be implemented and compared in the analysis section are the following:

4.1 Coordination

4.1.1 Global Planning

The first planning or coordination technique is based on performing a combined plan between every agent. To be more specific, this strategy is based on the Partial-Order Planning paradigm[26]. For this, it is needed to consider that agents with different perspectives of the environment will carry out a plan together.

The classic techniques for the planning consist on the knowledge of the world and on a list of actions that bring to the world from one initial state to the desired state. There are many techniques that define the actions in one way or another, such as refining over and over the plan, adding more actions to achieve the final state(Partial-Order Planning (POP)). Other strategies for instance replace the actions for fragments that will do those abstract actions (Hierarchical Task Planning (HTN)|27|).

The Partial-Order Planning is based on a set of actions that defines the plan. Therefore, this plan contains all the actions needed to reach the goal without an established order between them if it is not explicity necessary
(taking the key and opening the door are two different actions but the first one will be always before the second one). In this way, the agent can take decisions while the plan is being developed, satisfying the Principle of Least Commitment, which points out that the closest the decision taken process is to the execution of the decision, the highest chance of success.

The Partial-Order plan is composed by:
1. A set of actions (also known as operators).
2. A partial order for the actions: This is needed if there are actions that can only be executed after others.
3. A set of pre-conditions, conditions needed to perform an action.
4. A set of casual links that specifies which actions satisfy which pre-conditions.
5. A set of open preconditions, conditions that are not fulfilled by any action.

These techniques that have been optimized over time are usually oriented to only one agent, systems where there is only one agent that has to elaborate a plan. When it is focused in a multi-agent environment, the problem grows in complexity because the agents have to collaborate in order to achieve a common plan, but they can act selfishly because of their own goals. In fact, some techniques propose an environment with an absolute knowledge, which take us away from reality and real world, because in very few real agent systems the agents are able to know all the information.

The main problem when planning between several agents is how to coordinate them to establish the plan. From this perspective there are two main tendencies. Each agent can develop his own plan and merge it with the other’s plans or, they can create the plan together, coordinating the planning process. Most of the approaches are based on the first one, where there are several plans that, after merging them, a global plan is created that satisfies the global plan.

The decision to take this paradigm is because in Total-Order Planning approaches[26], if an incorrect action is taken, it is needed to introduce an
action that cancels the effects of the first one, and in the study case proposed in this paper, that is something impossible, since the environment depends on several aspects and agents do not know the exact results of their actions. On the other hand, the Partial-Order Planning paradigm is more flexible, since the actions do not have to follow an strict order.

Therefore, in this strategy, the agents will elaborate together a plan that will change by the agents refining it until they do not reach the final goal. In addition, the environment will have random events that make the refining of the plan more important.

![Figure 4.1: Global Planning Refinement Process](image)

Basically, every agent elaborates a plan. For this, an initial exchange of information is needed. After the elaboration the plan is sent to the other agent to evaluate and refine the plan, in order to make the new base plan, that will be refined in each iteration.

4.1.2 Structural Organizations

A structural organization[28] define roles, responsibilities and preferences for the agents within a cooperative society, as well as the control relationships and communications between them. From this global point of view,
the organizational structure associate to each agent the type of tasks that it can carry out, and priorities over some tasks. This structure increases the opportunities of success even if one of the agents fails.

The control relationships are those responsible of designing the authority of the agents and shape the different types of social interaction that can occur. There are two types:

1. **Hierarchical**: Only one resolver per hierarchy level have the authority for the decision-making and the control. The interaction is carried out between boss agents and subordinate agents. Boss agents have the control of resources and decision-making.
2. **Flat**: Each problem solver is a specialist in a specific area. Agents act through order and behaviour rules.

In the system there could be more than one specialist in the same area, but if there is only one type of specialists in the system, the flat structure cannot be used efficiently.

This structure simplifies the decisions that agents have to take, because they have fewer options (others will do some of those actions) and, therefore, their activities can be better directed. The main problem of this strategy is the decomposition of tasks in different roles.

### 4.1.3 Contractual Networks

This strategy[29] is an interaction protocol for cooperative troubleshooting between agents. Through this protocol, tasks can be assigned dynamically and make offers of different tasks in parallel.

The agent that offers the task is called manager. Their functions are the following:

1. To announce the task that needs to be carried out.
2. To receive and evaluate the offers.
3. To award the contract to a contractor.
4. To receive and synthesize the results.
On the other hand, the functions of the contractor are:

1. To receive tasks announces.
2. To evaluate its capacity to respond.
3. To respond.
4. To carry out the task if accepted.
5. To communicate the results.

All agents can play as a manager, making announcements of tasks.

This flexibility allows the task decomposition, since a contractor can play as a manager to solve his task, asking for help to other agents.

However, the limitations of the strategy are the following:
1. Manager has not the obligation of informing to the contractors that the task is adjudicated.
2. If there are specialized agents busy, the task can be assigned to a less trained contractor.
3. Similar to the previous one, important tasks can be not assigned because the contractors that can do it are working on other tasks.
4. Once the task is adjudicated, contractors cannot reject it.
5. Exhaustive communication is needed.

4.2 Negotiation

4.2.1 Social Election

In the Social Elections[30] strategies, agents give a common solution that will be voted. The result is forced and all the agents have to perform the decided solution.

It does not matter how the plans are obtained, only what agents do with the plans, the votation. This plan could be obtained from a leader agent, or from a list of plans for example. For this reason, this strategy could be merged with others, obtaining a good list of plans and voting them.
With this strategy, all the agents have an opinion and the aim is to find a common rule to make them all happy. The solution has to be pareto optimal.

### 4.2.2 Coalitions

When the agents have to negotiate, other option is to make coalitions\cite{31}. In those coalitions, some agents will be harmed, as they will have to do the tasks. In this way, the members of the coalition can coordinate the resources and decisions in order to increase the benefits.

The best way to implement this solution is creating a game in normal form and search the Nash Equilibrium in order to decide if the coalition helps the agents. If they agree, they solve the problem and distribute the benefits.

### 4.2.3 Task Assignment

This strategy is similar to the mechanism of Global Planning with the difference that each agent will be self-interested and will answer based on its own interests. This means that, although agents decide a common strategy, they can betray their partners and do different things.
Chapter 5

Implementation

Once the study-case is known and the methodologies to create a solution are explained, the next step is the selection of one of those methodologies, in order to implement the proposed system. First of all, the analysis and design phases of the methodology are applied. After that, the implementation can be developed, allowing results to be collected in order to analyze them.

5.1 Methodology

The methodology chosen is Gaia[21] because it has the enough complexity to develop a small multi-agent system from an initial set of requirements in a easy and fast way. Besides all the reasons stated in a previous chapter, one of the main reasons to choose this methodology is that it is focused on Roles that are clearly defined in the presentation of the problem. Given this particular scenario, the methodology fits perfectly with the problem.

5.1.1 Analysis

As it was mentioned before, the methodologies provide guidelines in order to achieve a right model of the system. According to the Gaia methodology, the first step is the analysis where the system is seen as a society. From this perspective, the programmer has to analyze all the possible roles, positions in the society. The following roles were obtained:
Roles Schemas:

After that, the roles proposed need to be defined. This definition helps the programmer to analyze if there is something missing in the system and which are actions that have to be done by other role. For this definition, the Gaia proposal is followed as it is shown in the next table:

<table>
<thead>
<tr>
<th>ROLES</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Rower</td>
<td>Rower Stats Controller</td>
</tr>
<tr>
<td>Referee</td>
<td>Navigator</td>
</tr>
<tr>
<td>Register</td>
<td>-</td>
</tr>
</tbody>
</table>

*Table 5.1: Roles*
Role Schema:

Rower

Description:

This role pushes the navigation object in order to arrive the final lane

Protocols and Activities:

Register, Sit, Row, ChangeSeat, Subscribe, ReceiveStart, ReceiveDockEntrance, ReceiveFinish, ReceivePitch, ElaboratePlan,

Permissions:

Reads:
- rowerStrength
- rowerResistance
- rowerAgility

Responsibilities:

Liveness - ROWER = (Register, Subscribe, ReceiveStart, ReceiveDockEntrance, Sit, (Row, ReceivePitch, ElaboratePlan, ChangeSeat)^n, ReceiveFinish)

Safety -

This schema is divided into:
1. **Role Schema**: The name of the role.
2. **Description**: What are the functions of the specified role.
3. **Protocols and Activities**: Interactions with other agents and private actions respectively. The activities are underline.
4. **Permissions**: The information available for the role.
5. **Responsibilities**: The functions of the Role expressed in a regular expression. The safety responsibilities show things that should not happen and that field can be empty.

Below, the rest of tables are shown:

<table>
<thead>
<tr>
<th>Role Schema:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rower Stats Controller</td>
</tr>
</tbody>
</table>

**Description:**

This role checks the stats of the rower when a new row is done, calculating the results of the row and informing the navigator system.

**Protocols and Activities:**

GetRow, CalculateVector, InformRow,

**Permissions:**

Reads:
- rowerStrength
- rowerResistance
- rowerAgility

**Responsibilities:**

\[ \text{Liveness} - \text{ROWER STATS CONTROLLER} = (\text{GetRow}, \text{CalculateVector}, \text{InformRow})^n \]
Safety - rowerTiredness <= rowerStrength
5. Implementation

<table>
<thead>
<tr>
<th>Role Schema:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigator</td>
</tr>
</tbody>
</table>

**Description:**

This role moves the rowers from the starting line to the finish line

**Protocols and Activities:**

ChangeSeat, PositionChange, NewRow, CalculateBoatPosition

**Permissions:**

Reads:
- rowerSeats
- boatPosition
- headingDegrees

Writes:
- rowerSeats
- boatPosition

**Responsibilities:**

Liveness - NAVIGATOR = ( (ChangeSeat)*,(NewRow, CalculateBoatPosition, PositionChange)* )^n
Safety - headingDegrees < 180 headingDegrees > -180
### Role Schema:

Referee

#### Description:

The referee is in charge of giving information about the boat situation to the rower

#### Protocols and Activities:

ReceiveBoatPosition, SubscribeRowers, ReceiveTeams, StartRace, InformRowers, FinishRace,

#### Permissions:

Reads:
- circuitLength
- boardingExtension

#### Responsibilities:

Liveness - REFEREE = (ReceiveTeams, SubscribeRowers, StartRace, (ReceiveBoatPosition, InformRowers)\(^n\), FinishRace)
Safety -
### Role Schema:

**Register**

**Description:**

Register the rowers in the race

**Protocols and Activities:**

RegisterRower, SendRowersReferee

**Permissions:**

*Reads:*
  - registrationTimeoutPeriod

**Responsibilities:**

Liveness - REGISTER = (RegisterRower, SendRowersReferee)

Safety -

---

#### 5.1.2 Design

The roles schemas provide a preliminary description of the entire system. This is a representation on how the programmer thinks and there is not any tool that ensures correctness. From these schemas, it is possible to start the designing step based on the requirements or scenario, where the following models are created:

**Agent Model:**

In this model, the real agents are defined. Once the roles are created, the programmer has to assign them to the agents. Again, the programmer can decide to have one agent per role or, if the roles are similar, to assign more than one role to the same agent. It also depends on how the roles schemas were created.
5. Implementation

Rower - Rower (4)
Rower Stats Controller - Fatigue Control (1)
Referee - Referee (1)
Navigator - Boat (1)
Register - Registration Desk (1)

Acquaintance Model:

After the creation of the agents, the next step is to define the communication paths. In this model, the programmer points out the different possible communications in the system.

![Acquaintance Model Diagram]

Figure 5.1: Acquaintance Model

Services Model:

Finally, the service model is created. This model can be defined together with the implementation. The services are based on the communications and activities/protocols. When the programmer is coding all these communications, the different inputs, outputs and conditions come up. This model is the one at the lowest level, since it is possible to see real variables used in the code. If it is created together with the implementation, every time the programmer codes a new communication, a new row is added to this model, creating the services model.
### Table 5.2: Services Model 1

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Seat</td>
<td>TargetSeat</td>
<td>DecisionResult</td>
</tr>
<tr>
<td>Register</td>
<td>RegistrationForm</td>
<td>DecisionResult</td>
</tr>
<tr>
<td>Sit</td>
<td>TargetSeat</td>
<td>DecisionResult</td>
</tr>
<tr>
<td>Row</td>
<td>RowInputs</td>
<td>Tireness, TimeRow</td>
</tr>
<tr>
<td>Subscribe</td>
<td>BoatId</td>
<td>DecisionResult</td>
</tr>
<tr>
<td>Boat Position Change</td>
<td>Position</td>
<td></td>
</tr>
<tr>
<td>Send Race Start</td>
<td>raceStarted</td>
<td></td>
</tr>
<tr>
<td>Send Race Finish</td>
<td>raceFinished</td>
<td></td>
</tr>
<tr>
<td>Send Dock Information</td>
<td>Position</td>
<td></td>
</tr>
<tr>
<td>Send Teams to referee</td>
<td>QualifiedTeams</td>
<td></td>
</tr>
<tr>
<td>Inform Plan</td>
<td>RowerPosition</td>
<td></td>
</tr>
</tbody>
</table>

### Table 5.3: Services Model 2

<table>
<thead>
<tr>
<th>Service</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Seat</td>
<td>toChange &amp;&amp; onBoat</td>
<td>!toChange &amp;&amp; !onBoat</td>
</tr>
<tr>
<td>Register</td>
<td>!isRegistered</td>
<td>isRegistered</td>
</tr>
<tr>
<td>Sit</td>
<td>!isSit &amp;&amp; onBoat</td>
<td>isSit</td>
</tr>
<tr>
<td>Row</td>
<td>currentTime - lastTimeRow &gt; timeRow</td>
<td>lastTimeRow = currentTime</td>
</tr>
<tr>
<td>Subscribe</td>
<td>raceAccepted</td>
<td></td>
</tr>
<tr>
<td>Boat Position Change</td>
<td>posChange</td>
<td>!posChange</td>
</tr>
<tr>
<td>Send Race Start</td>
<td>isStart</td>
<td>!isStart</td>
</tr>
<tr>
<td>Send Race Finish</td>
<td>isFinish</td>
<td>!isFinish</td>
</tr>
<tr>
<td>Send Dock Information</td>
<td>isRunning</td>
<td>!isRunning</td>
</tr>
<tr>
<td>Send Teams to referee</td>
<td>currentTime - startRegTime &gt; regTime &amp;&amp; !informed</td>
<td>informed</td>
</tr>
<tr>
<td>Inform Plan</td>
<td>informOthers</td>
<td>!informOthers</td>
</tr>
</tbody>
</table>

### 5.1.3 Models Implementation

With these models and schemas developed following the Gaia methodology, all the information is presented, so it is easy to create an implementation according to them. The models support enough information to know where everything is in the system, but it still leaves a lot of freedom to the designer.

For that reason, in this section, each role implemented is briefly explained.

The system is basically composed by the roles classes mentioned above.
Each role has different behaviours that allow it to perform the correct activity every moment. Those behaviours are isolated from the main class for the purpose of doing the system more readable. Roles have a lot of internal information in order to know which behaviour is active. This causes the existence of more than 55 classes in the system, counting the agent behaviours, together with 35 classes that represent the ontology needed to perform the communications. All the communications are based on FIPA standars, using the JADE framework(Appendix A). More information about the communication protocol can be saw in the appendices.

Rower

The rower is the main class of the system but also the most important class for the purpose of the thesis. In fact, all the other agents are not needed, since the study is based on the techniques used between rowers; but in order to create a real system, to simulate the reality and obtain realistic data, a system with variety of different roles must be created.

The rower starts registering to the race, informing the registration desk about his team number. After this step, the agent has to wait to see if the registration desk accepts. If the team is accepted, the rower gets prepared for the race by subscribing to different events offered by the referee. The register and subscription is done using the ServiceDescription, that Works as yellow pages.

Rowers do not know anything about the boat, only the pitch is given by the boat so they need somewhere to get all the information. That is the reason why they subscribe to the referee. During the race, the rower receives messages from the referee about the start, current position and end of the race.

After the initiation of the race, the rower waits for the receiving of the message that indicates that the boat is in the rower’s dock, allowing the agent to jump into the boat. Once the rower is in the boat, the next step is to sit and start rowing. During the rowing, the agent sometimes has to change its position, in order to fix the pitch. This step depends on the strategy adopted by the system and is explained in the section about strategies implementation.
Finally, rowers arrive the final goal, finishing their execution.

**Registration Desk**

The Registration Desk class is one of the simplest classes as it only has two behaviours. In the first one, the Registration Desk is registered in the yellow pages, so agents can search for him. From that moment, the agent waits for ten seconds. During those seconds, the Registration Desk receives petitions of rowers in order to register to the race. If four rowers of the same team are registered, the Registration Desk accepts the team. If not, a refuse notification is sent to them. The second behaviour only needs to send to the referee and to the boat the information about the rowers that are currently registered and accepted to participate the race.

**Boat**

The boat is in charge of carrying the rowers from the initial line to the final line. The boat is created only after the acceptance of the team in the race. In that moment, the registration desk creates one boat per team participating. From that moment, the boat is waiting for rowing messages. Those messages have to pass through the fatigue control, which ensures that the rower is doing the rows correctly. When the boat receives those messages, the rowing vector is extracted and all the operations presented in the chapter 2 are calculated. With these results, the boat can simulate a real movement through the water. There are several components in these calculation, for example, it is not the same a row when the boat is immobile than when the boat is moving due to a previous row.

After the row is calculated, the boat informs the referee about the new position of itself. This data goes to the referee because that agent is the one having the information about the environment. In a real environment, there are always visual signals or a speaker giving the information to the rowers since the rowers cannot know that data with precision if they are inside the boat.

In addition, the boat can receive petitions to change the seat of a rower. This communication is done directly between the rower and the boat.
Fatigue Control

The Fatigue control only needs to ensure that boats are receiving the correct information. In a simulation, this agent could be removed, but the problem presented imitates a real environment. In that situation, more than one team can be rowing and if they send directly the data of the rows to the boat, they can cheat. On the other hand, if there is an intermediary that translates the rows, it is impossible to trick the system. The intermediary receives the position of the rower, the oars and the intensity that the rower wants to use. These data is converted to a vector that is sent to the boat.

This intermediary could be an object oriented class, but defined as an agent the system is more consistent and the rowers maintain the structure of the messages when rowing.

Referee

Finally, the referee is the one with a global vision of the environment. This agent has three basic responsibilities. Inform about the start of the race, the current position of the boats an end of the race.

As mentioned above, when rowers are registered, the registration desk sends the teams information to the referee. These teams can subscribe to the three events the referee offers. When the race starts, the referee iterates the list of rowers subscribed to that event and send an inform message to each of them. When the boat informs the referee about its new position, the referee iterates a list again, in this case the list of rowers subscribed to the event that informs of new boat positions. Finally, the same happens when the race is over with the corresponding list.

Rowers always need to subscribe to those events to work properly, especially the one that gives the boat positions, since the rowers do not start inside the boat and they need to enter when the boat is in their dock.

Low-level examples

In order to show how JADE works, a few examples are showed below.

First of all, a picture with an agent action is showed.
Figure 5.2: Agent Action

The actions are part of the ontology, that defines the entire domain in order to allow communications between agents, including services (agent actions) and the concepts (terms) contained in the services. The next image represents a concept:
These concepts have to be sent when the agent tries to communicate with others, but first the agent have to be defined. The agent also needs to know the ontology:

```java
public class RegistrationForm implements Concept {

    /**
     * Protege name: dock
     */
    private Dock dock;
    public void setDock(Dock value) {
        this.dock=value;
    }
    public Dock getDock() {
        return this.dock;
    }

    /**
     * Protege name: teamId
     */
    private TeamId teamId;
    public void setTeamId(TeamId value) {
        this.teamId=value;
    }
    public TeamId getTeamId() {
        return this.teamId;
    }
}
```

Figure 5.3: Agent concepts
In the agent class, all the variables needed to carry out their functions are established. One of those variables are the ontology, obtained from the Ontology class. In addition, the behaviours are added. In those behaviours, the communications are performed:

```java
public class Rover extends Agent {

    private PropertyLoader properties;
    private Codec codec = new SICodec();
    private Ontology ontology = Ontology.getInstance();
    private boolean toChange = false;
    private boolean subscribed = false;
    private boolean onBoat = false;
    [...]

    protected void setup() {
        addBehaviour(new RegisterBehaviour());
        addBehaviour(new SitBehaviour());
        addBehaviour(new RefuseBehaviour());

        // Load service names
        properties = PropertyLoader.getInstance();
    }
```

**Figure 5.4:** Agent Definition
In the behaviour presented above, the rower tries to find the Registration Desk in the yellow page service (DFAgentDescription). If the agent exists, the next step is to create the message asking for the registration. There are different types of messages offered by JADE, choosing for this specific case the request. Finally, all the concepts needed for the services “Registration” are fulfilled in order to send the message.

In case of the reception, the process is very similar:
In this case, the boat is being asked for a seat change. The boat waits until a new message with the known ontology, protocol (service) and type of message arrives. After the reception, the agent extracts the content in order to perform the actions required.

5.2 Strategies

With the high details of the system created, rowers’ strategies can be implemented, collecting the results of each one. With these results, it is possible to do the empirical study of each strategy and understand, with real data, the advantages and disadvantages of each one.

5.2.1 Coordination

Global Planning

As mentioned above, this strategy is based on:

1. Making an initial plan that will change each iteration.
2. Seeing if the subgoals are achieved. In that case, a new initial plan is set.
3. Checking if the final goal is reached, obtaining the solution.
4. If not, redo the process in a new refinement of the plan.

With this steps presented, in the following paragraphs the implementation is explained:

First of all, each rower will elaborate his own plan. Their goal is to reach the goal line while pushing the boat, so the plan is divided in several pushes.
5. Implementation

Every iteration, rowers will study if the boat is deviating, that is, if the pitch is the correct one or not. To correct this deviation is the first subgoal of the agent and, in order to correct it, the rowers will have to move from one seat to another, rowing stronger to one side and following a plan elaboration.

In the first step of the plan elaboration, the rower studies the position of the rowers compared to the pitch deviation. To do that, the rower looks at the position of each rower and adds their value to a counter. This value is calculated using their tiredness and a position factor as explained in the next image:

![Figure 5.7: Seats coefficients](image)

This counter will have the current overall strength and direction of the boat. For instance, if the result is 0, that means that the boat is going straight.

Then, the pitch is changed to a number between -2 and 2 following the next criteria:

1. If \( \text{pitch} < -45^\circ \) then \( \text{newPitch} = -2 \)
2. If \(-45^\circ < \text{pitch} < 0^\circ \) then \( \text{newPitch} = -1 \)
3. If \( \text{pitch} = 0 \) then \( \text{newPitch} = 0 \)
4. If \( 0^\circ < \text{pitch} < 45^\circ \) then \( \text{newPitch} = 1 \)
5. Implementation

5. If pitch > 45° then newPitch = 2

When the newPitch is calculated, it is added to the overall strength/direction of the boat. The result is again limited to the range (-2,2). This final result gives the number of movements needed to correct the direction.

For instance, this can be saw easily in a particular case:
When the overall strength and direction, called Strength, is calculated, the result is 1. The newPitch gives a result of -1, which means that the boat is rotating to the left and need to be corrected.

Therefore, the next step is to add those two values. In that case the result is 0, so no movements are needed. This is because, although the boat is rotating to the left, the rowers are pushing stronger to the right. This is probably because the pitch was tried to correct in the previous plan.

Now, in the case of a Strength of 0, the final result would be -1. Then the rower tries to create a plan where one rower moves one place to the right. If the final result is -2, one rower have to move two places to the right, or two rowers one place to the right each one.

With this algorithm, the new plan is elaborated.

Every time the rowers refine the plan because of the deviation, each rower will send his proposal to the others. In this manner, rowers will receive as many plans as agents in the boat, and from the list of received plans, the rower will choose the one that fits better according to his knowledge. At this point the first problem is showed.

In this approach, all the agents have similar conditions and all of them will elaborate the same plan. This means that the global planning is useless as they do not have to merge the solutions. This situation detracts the value of the study. Because of that, each rower assigns a random tiredness value to his partners. With this approach, each rower will develop the plan according to their knowledge, deciding together which plan to carry out.

Therefore, rowers share the plans. The selection of the plan is up to the programmer so it will be decided by voting. The plan with more votes will be selected, as it is the plan that satisfies more rowers. If the vote ends in
a tie, one rower has to take the leadership and decide the plan. Obviously, rowers cannot choose their own plan.

Once they have select a plan that satisfies the subgoal (correcting the deviation), rowers have to check if they have reach the final line. If not, a new iteration will be carried out, refining again the plan.

These strategy will return a list of actions, movements that allows the rowers, from a initial state and position, reaching the goal, finishing the race.

With few agents, it is difficult to obtain a majority. Every time a rower enters the boat and increases the number of 'experts', a better solution is obtained, since with their knowledge, their own plans will be always the best one.

**Structural Organizations**

The basis of the functionality of this strategy is similar to the Global Planning strategy. In fact, all the strategies have the same common basis, since all of them have to generate a plan to act according to it. The plan generation is the same explained above. The steps needed are:

1. To select a boss between the rowers.
2. The boss creates the plan.
3. The boss send orders to the other rowers.

The main difference in this strategy is that only one of the agents decides the plan. There are a boss and their subordinates. That rower decides the plan to do and the other agents do it. It is very similar to the previous one, skipping the merge step.

This solution is centralized and it could create a bottle neck, but if the boss has enough knowledge from the others, the solution will be more optimized.
Contractual Networks

In this strategy, there is again only one agent who decides the plan to follow. The difference is that the agent does not select the rowers that will move. The manager asks the contractors for a movement to a specific place. The rowers will offer their services, saying their characteristics so the manager can choose the best one, making a contract with him:

1. To select the manager.
2. The manager talks with the contractors, asking them their tiredness.
3. The manager order the rower selected to move to a different position if needed.
4. The rower changes its position.

5.2.2 Negotiation

Social Election

The Social Election is also very similar to the Global Planning, since every agent make a plan that is offered to the other rowers and a voting process begins. The difference here is the way to make the plan. In the previous cases agents were looking for the best solutions but now, there is also a private goal. For that reason, rowers will not only try to find the best solution, but they will also try to do the less possible movements, choosing plans where other agents do the movement. So, basically the plan elaboration is the same, but the last step, where rowers are chosen to do the movement, the rest of the rowers are chosen first and, only if there are not more rowers available, the rower is chosen itself. The process can be described as:

1. To obtain a list of plans. In this specific case, each rower will elaborate a plan, prioritizing other’s movements.
2. All the agents vote trying to find an agreement.
3. Rowers change their seats if needed.

Coalitions

In this strategy, rowers can make coalitions, forcing one rower to do the movements by voting the same result. Instead of searching for the best solu-
5. Implementation

tion, the members of the coalition search for a solution where the movement is done by a specific agent:

1. The coalition is formed. In this case, rowers 1 and 2 will make a coalition.
2. Rowers elaborate their own plan. In the case of a member of the coalition, it will focus on others movements, avoiding the movements of its ally.
3. The plan is selected through a votation.
4. Rowers move if needed.

**Task Assignation**

The last negotiation technique will apply the Global Planning strategy, making a common plan but, as they do not have to obey, two different scenarios will be presented:

1. Two rowers will never rely on the other agents, so they will never change their seat.
2. The four rowers will never rely on the other agents, so they will never change their seat.

It can be also defined with the following steps:

1. Rowers create a new plan, following the steps described in the global planning.
2. Rowers select the new plan.
3. Rowers 1 and 2 will move if needed and rowers 3 and 4 will never move in the first case. In the second case no one of them will move.
Chapter 6

Analysis

Finally, with the empirical data obtained, the following chapter will make an analysis of the strategies implementations.

As explained in the introduction, scalability, efficiency and Robustness attributes were studied. Each strategy was executed 50-60 times, 25-30 of which were done with random events, such as random changes on the orientation of the boat. From these executions, the following data were obtained:

1. Maximum time needed, the worst case.
2. Minimum time needed, the best case.
3. Average time of all executions.
4. Number of messages between the rowers and other agents.
5. Number of movements of each rower.

With the times, efficiency will be studied. The number of messages describes the scalability of the system. There is a relationship between the number of messages and movements.

6.1 Results

After the application of the implementations suggested, the following results were obtained:
6. Analysis

6.1.1 Coordination

<table>
<thead>
<tr>
<th></th>
<th>Structural Org.</th>
<th>Global Planning</th>
<th>Contract. Networks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Time</td>
<td>73.06 s</td>
<td>69.13</td>
<td>60.19</td>
</tr>
<tr>
<td>Min Time</td>
<td>60.21</td>
<td>64.28</td>
<td>57.97</td>
</tr>
<tr>
<td>Average Time</td>
<td>69.27</td>
<td>66.79</td>
<td>59.23</td>
</tr>
<tr>
<td>Number of Msg</td>
<td>451</td>
<td>609</td>
<td>627</td>
</tr>
<tr>
<td>Rower 1 movements</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Rower 2 movements</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Rower 3 movements</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Rower 4 movements</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.1: Non-Random Coordination Results

<table>
<thead>
<tr>
<th></th>
<th>Structural Org.</th>
<th>Global Planning</th>
<th>Contract. Networks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Time</td>
<td>81.42</td>
<td>79.08</td>
<td>65.31</td>
</tr>
<tr>
<td>Min Time</td>
<td>72.41</td>
<td>68.48</td>
<td>60.43</td>
</tr>
<tr>
<td>Average Time</td>
<td>76.17</td>
<td>74.10</td>
<td>63.28</td>
</tr>
<tr>
<td>Number of Msg</td>
<td>585</td>
<td>812</td>
<td>847</td>
</tr>
<tr>
<td>Rower 1 movements</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Rower 2 movements</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Rower 3 movements</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Rower 4 movements</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.2: Random Coordination Results

Studying those data, some results can be observed:

In terms of efficiency, the contractual networks are clearly the fastest. The average is even better than the best results obtained by the other approaches. This is because the contractual network always choose the best option between the available agents. In this case, all the agents are always available so the manager can ask to the most optimal agent, but, if the number of seat changes is observed, this approach is the one that needs more movements. The explanation for this happening is that, while in the other strategies agents have a false idea about the tiredness of the agents, in the contractual networks the manager knows the exactly tiredness of each agent.

When agents are more or less equal tired, as agents think in the global planning and structural organizations, a movement does not change too
much the situation; but if one agent is rested and the others very tired, (the manager in the contractual network knows that) if they interchange their positions, the improvement can be very important.

For that reason, contractual networks are the most efficient in terms of time, but that strategy needs more resources in terms of movements.

Between the Global Planning and the Structural Organization, the Global Planning average is fastest because, although none of them knows their partners tiredness, a decision taken by several members tends to be more accurate than the decision taken by one. A curious result found in this aspect is that the minimum time needed is better in the Structural Organization. This could be because when the decision is taken by more than one agent, the error is minimized, but when a leader takes the decisions, the results range is bigger, together with the possible error.

About the quality attribute scalability, the most important data to study is the number of messages. For this particular case, only the messages between agents were counted. In this system there are few agents, but they interchange a lot of messages. If the system is bigger, these messages can crash the system. JADE is very useful in that sense and can handle a great number of messages, but also the strategy is very important. In this situation, it is visible the advantages of having a leader coordinating the team. When agents have to decide a common plan, the process of refining the plan takes a lot of time. Also, if the manager has to ask the attributes of each rower, the number of messages is elevate. On the other hand, if only one agent decides what to do, that agent only needs to send one message (and the acceptance response) to each rower, communicating its plans. In the following chart shows how the system scales depending on the number of rowers:
Figure 6.1: Number of messages between rowers per row

The chart shows a big difference between global planning and the others. In previous chapters, it is mentioned that the structural organization centralizes the decision-making, creating a bottle neck. The advantage of this situation can be seen in this chart. In the results, the contractual network has worst results than global planning because the number of movements, but if the number of rowers is increased, contractual networks will have a better performance.

The formulas that define the different strategies are:

\[
\text{GlobalPlaning} = 2 \times n \times (n - 1) \\
\text{StructuralOrganization} = n - 1 \\
\text{ContractualNetwork} = 2 \times (n - 1) + 1
\]

The last quality attribute that is clearly different depending on the strategy is the robustness. Only in the case of the global planning, the death of an agent does not change the results. If the manager in the contractual network or the leader in the structural organization dies, the rest of rowers will not know what to do, and they will not reach the goal.

Other factor to take into account is the number of times in which the agents were too exhausted to finish the race. This data is not in the tables.
as it is out of the scope of the thesis, due to the computational resources needed to calculate a significant number of instances to see how many times the systems ends successfully. In the case of random events, sometimes, with the pitch changing continuously, the agents did not finish the race.

6.1.2 Negotiation

<table>
<thead>
<tr>
<th></th>
<th>Social Election</th>
<th>Coalitions</th>
<th>Two Non-Trust</th>
<th>Four Non-Trust</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Max Time</strong></td>
<td>70.52</td>
<td>87.48</td>
<td>200.83</td>
<td>-</td>
</tr>
<tr>
<td><strong>Min Time</strong></td>
<td>63.98</td>
<td>73.57</td>
<td>190.06</td>
<td>-</td>
</tr>
<tr>
<td><strong>Average Time</strong></td>
<td>67.12</td>
<td>78.32</td>
<td>195.27</td>
<td>-</td>
</tr>
<tr>
<td><strong>Number of Msg</strong></td>
<td>634</td>
<td>723</td>
<td>1766</td>
<td>-</td>
</tr>
<tr>
<td>Rower 1 movements</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Rower 2 movements</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Rower 3 movements</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rower 4 movements</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6.3: Non-Random Negotiation Results

<table>
<thead>
<tr>
<th></th>
<th>Social Election</th>
<th>Coalitions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Max Time</strong></td>
<td>80.31</td>
<td>97.91</td>
</tr>
<tr>
<td><strong>Min Time</strong></td>
<td>69.22</td>
<td>79.11</td>
</tr>
<tr>
<td><strong>Average Time</strong></td>
<td>77.83</td>
<td>92.32</td>
</tr>
<tr>
<td><strong>Number of Msg</strong></td>
<td>845</td>
<td>971</td>
</tr>
<tr>
<td>Rower 1 movements</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Rower 2 movements</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Rower 3 movements</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Rower 4 movements</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6.4: Random Negotiation Results

In the case of the negotiation, results are much clearer. As the agents want a common goal, the competition in order to satisfy own goals harms the final result.

The efficiency represents the biggest differences between the strategies, only the social election gives good results. This is because the pattern followed is exactly the same than in the global planning. The only difference is that the rowers try to avoid their own seat change when they are making
the plan, but at the end, they vote the plans and the plan chosen is a good plan.
In the coalition, agents forget the main goal, forcing one rower to do most of the seat changes. This is not a good choice because when the agent starts to get tired, it is more difficult to correct the pitch and making the boat to do a zigzag movement. With random events, the results are exactly the same. Finally, the last two strategies follow the “never-trust” patterns. In the first place, two rowers never change their seat. This means that if the agents create a plan where those agents have to move, they will stay in their positions. The result of this strategy is that the other two rowers have to do extra movements, trying to correct the previous plan because when the boat is losing the correct heading, it can be easily correct with a movement, but if that movement is not done, the boat will start to move to the starting line and the correction of the pitch will need more time, if they are able to do it (most of times, they do not reach the goal).
In the second strategy, all the agents refuse to move. As expected, all the times this strategy was executed, the execution failed.

Nevertheless, in a competitive system where the number of seat changes are more important than time, the coalition strategy gives the best results for some of the rowers. Also the non-trustable rowers obtained a very good result, since they did not do anything and sometimes they reach the goal.

About the scalability, again the social election is the best. The explanation is the same. With a straight motion, rowers reach earlier the goal, they need less time and seat movements, which means less messages.

Finally, in terms of robustness, all the strategies are similar. If one agent dies, the system can continue working. In fact, sometimes the death of an agent in the last cases gives better results, since the death of a non-trustable agent means less misunderstandings.

**Coordination vs Negotiation**

In this particular case, it was obvious before execution that the coordination techniques will give better results, as all the rowers have a main goal. If rowers are selfish and try to achieve their own goals, the common goal will be affected, as can be seen in the results. Only the social election gives similar results, but in that strategy, if the personal results are observed, agents did
not get better results. The way to decide the plan is different, but it is still a common plan. In fact, one execution was better than the global planning strategy.
Chapter 7

Conclusions and Future Work

7.1 Conclusions

The following conclusions have been derived from the empirical study:

1. First of all, it is important to point the big difficulties to apply coordination techniques in a multi-agent system with more than two agents with the same goal. Most of the existing strategies follow a process of creating and refining of a plan. This situation is easy to handle when two agents have a conversation. One of them creates the plan that sends to the other one, which can study the plan and refine it. This situation is repeated until a final plan, where both agents are satisfied, is reached; but when there are lots of agents deciding, or they follow an order, that is less consistent and robust, or they create multiple concurrent refinements of the plan.

2. The second main conclusion is the certainty that there is not a perfect technique for every situation. Depending on the pursued objectives, a different strategy is better. In this particular case, the results show that in terms efficiency, the technique with more knowledge will make the best plan. In this experiment, it is assumed that in the Global Planning strategy, agents do not have knowledge about the other rowers. If this is changed, the results obtained will be similar to the structural organization, since all the agents are exactly equal implemented. As mentioned in previous chapters, this decision was taken in order to test that technique, because if they are equal implemented, the merging
and refinement could be skipped.

3. In addition, results are very clear about the benefits of coordination techniques compared to negotiation techniques when there is a common goal. Also, when more agents take the decisions, the range error is decreased.

4. Finally, it is worthy to remark the great importance of the time and number of messages. In the Non-Random scenarios, where everything should work in the same way each execution, the results obtained are very different. In multi-agent systems the environment is always changing and the agents react to those changes, and, as there is a simulation, sometimes the computer needs more time to deliver a message. In those cases, the environment is still changing. In this example, the boat is still moving and for that reason, every simulation is different.

7.2 Future Work

The two main approaches for future work are:

1. The study of more strategies together with more executions. Because of the time of the thesis, only 6 strategies could be studied.

2. To perform a more exhaustive analysis with more attributes and data collected. With more executions it is easier to take conclusions related to specific quality attributes.
Bibliography


Appendix A

A.1 Java Agent DEvelopment Framework (JADE)

JADE (Java Agent DEvelopment Framework) is a software Framework fully implemented in Java language.

JADE is an open source software under LGPL license. JADE agents platform is composed by an agent management system (AMS) and a yellow pages service (DF). They communicate through a message transport system. There are more optional agents in order to monitor the system, seeing the messages send through the framework, such as the Introspector Agent. JADE also gives a graphical interface to deal with these agents and a library of FIPA interaction protocols (The JADE Agent Platform complies with FIPA specifications). There is a plugin for Eclipse called EJADE that facilitates the use of JADE.

Agents are able to accomplish several tasks simultaneously, but every JADE agent have a unique execution thread, so tasks are changing continuously. These tasks are implemented as behaviours, which during the execution are added and removed. Each behaviour has four basic methods: onStart(), action(), done() and onEnd() that describes how the agent works.

A behaviour can be blocked waiting for a message. In that case, the agent can wake up when a message arrives. Each rower has a FIFO queue of messages but they can search for specific messages specifying message templates such as the performative (Inform, Accept), the ontology or the protocol of a specific ontology.

The life cycle of an agent in JADE is the following:
Figure A.1: JADE Agent life cycle