FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Multi-agent system for simulation and validation of scenarios

João Pedro Correia dos Reis

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Gil Manuel Magalhães de Andrade Gonçalves (Professor)

Second Supervisor: Paulo Dias (Investigator)

July 30, 2012

Multi-agent system for simulation and validation of scenarios

João Pedro Correia dos Reis

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Doctor Rosaldo José Fernandes Rossetti
External Examiner: Doctor Luís Miguel Martins Nunes
Supervisor: Master Gil Manuel Magalhães de Andrade Gonçalves
July 30, 2012

Abstract

The goal of this dissertation is to study the applicability of multi-agent systems in a military simulations context using the development of an agent-based platform for simulation and validation. Given a certain configuration – positioning, characteristics and dynamics of assets – it's important evaluate its effectiveness in protecting a certain region or infrastructure, knowing the communication protocols and strategies that represent the real life protection.

This work has been developed within the scope of the SAFEPORT project, currently in execution under the "Defence Against Terrorism Program of Work" (DAT-POW) of NATO, which aims to develop a Decision Support System to aid in the definition of the best strategies for the protection of harbors or expeditionary fleets. This DSS is composed of several independent modules with distinct functions, in which a modular communication is a requirement.

The multi-agent platform developed in this dissertation is part of this overall system and aims in providing information about the proposed strategies (in terms of pre-defined key performance indicators) based on simulation results. The result of the simulation (several simulation runs) represents an assessment of a certain defense configuration applied in the protection of a given infrastructure or fleet, using a set of resources and strategies.

One of the main motivations of this project is to give the possibility for the user to evaluate the proposed maritime assets configuration and its corresponding behaviors for surveillance and protection of a certain region, without costs and life-threatening conditions. The applicability of an agent-based approach is also a motivation for the project development, being the implementation of multi-agent systems for simulation in military systems an area in expansion.

Resumo

Esta dissertação tem como objetivo o desenvolvimento de uma plataforma multiagente de simulação e validação, e o estudo da aplicabilidade de sistemas baseados em agentes num contexto de simulação. Dada uma determinada configuração - disposição de meios marítimos – é importante saber se esta é ou não eficaz na defesa de uma determinada infraestrutura, tendo conhecimento dos comportamentos dos meios disponíveis e respetivas formas de comunicação. Esta dissertação enquadra-se no projeto SafePort proposto pela NATO, fazendo assim parte de um plano mais alargado constituído por diferentes módulos independentes, com funções distintas, sendo requisito uma comunição modular.

O objetivo do projeto de dissertação passa por fornecer informação relativamente aos resultados da simulação, representando uma validação de uma dada configuração dos meios militares disponíveis para a defesa de uma dada infraestrutura, bem como as suas estratégias associadas. Para isso foram utilizados os conceitos explorados na revisão bibliográfica, simulando um ambiente hostil através da informação modelada sobre os comportamentos e estratégias dos meios.

Uma das grandes motivações para a realização desta dissertação passa por dar a possibilidade aos utilizadores de validar os meios aplicados num ambiente de vigilância e proteção, e os seus respetivos comportamentos, sem que custos e ameaças de vida humana sejam uma possibildiade. A utilização de sistemas de simulação e validação baseada em agentes representa também uma motivação ao desenvolvimento deste projeto, visto ser uma área em expansão dentro da simulação computacional, e mais especificamente na simulação militar.

No desenvolvimento do projeto, foram aplicados alguns processos no que toca à implementação de um sistema de simulação, e na modelação correta de um comportamento ou de uma estratégia, bem como tecnologias que permitem a correcta comunicação modelar - modelo de simulação - e modular - todos os módulos que participam no projeto na sua generalidade. Estes processos são fruto do estudo já realizado sobre as áreas de contexto da dissertação, sendo adaptadas aos requisitos do sistema para a realização da mesma.

Acknowledgements

The writing of this dissertation has been one of the most enriching and rewarding experiences regarding the requirement of independent learning given the project context, being undoubtedly the most important and significant academic challenge. Without the support, patience and guidance of the following people, this study would not have been completed, and it is to them that I owe my deepest gratitude.

- Professor Gil Gonçalves who undertook to act as my supervisor, being the most important guidance in both project implementation and dissertation issues;
- Researcher Paulo Dias, who undertook to act as my co-supervisor, which provided all the necessary military knowledge for the project implementation, and exposed the whole project in the NATO Research and Technology Agency, when it was in development;
- To all my friends that supported me and inspired in all the efforts made along this academic journey.

João Pedro Correia dos Reis

"We have to remember that what we observe is not nature in itself,
but nature exposed to our method of questioning."

Werner Heisenberg

Contents

List of Figures

LIST OF FIGURES

List of Tables

Abbreviations

ABEL	Advanced Boolean Expression Language
ACL	Area Command Level
AMS	Agent Management System
BDI	Belief Desire Intention
DFA	Deterministic Finite Automata
DS	Directory Facilitator
DSS	Decision Support System
FIPA	Foundation for Intelligent Physical Agents
GADT	Global Agent Descriptor Table
IMTP	Internal Message Transport Protocol
KPI	Key Performance Indicators
MAA	Multi-Agent Architecture
MAS	Multi-Agent System
M&S	Modeling and Simulation
NATO	North Atlantic Treaty Organization
NCL	National Command Level
PLD	Programmable Logic Device
RSAC	Rand Strategy Assessment Center
XML	eXtensible Markup Language

Chapter 1

Introduction

Regarding the military context, one of the most common issues refers to the managing and appliance of aerial, land and maritime available means, in both strike and defensive environments. For the problem solving of this approaches, differential equations were explored in the early of twentieth century, by *F. W. Lanchester and Epstein* [Smi98]. Those equations were a major step for the modeling and simulation process in military warfare, being nowadays used for the simulation of several kinds of strategies in different means in hostile situations [Ila04].

Taking into account the past two decades, military simulation systems have been changing its paradigm and the way they are implemented computationally. Those paradigms refers to artificial intelligence usage, in which was a changing of simulation approach, passing from differential equations to an individual and decentralized approach, allowing this way an adaptive behavior modeling [Ila04]. The usage of multi-agent architecture regarding simulation, modeling and validation, has becoming to prove adequate for both individual and global strategies and behaviors. The study of complex dynamic systems modeling has been explored, being one of the most important contexts for the diverse agents' implementation and heterogeneous behavior.

This kind of problem regarding the simulation context has a major influence in the military scope, since there are several combat means distributed around the world, with the purpose of harbor surveillance and defense. The agent-based simulation study allows the validation of strategies that every nation has about their available defense and strike means. This document has the main purpose to present not only the implemented project in the dissertation scope, but also a discussion of the agent-based simulation regarding the military warfare, and show some work that were performed in the area of Distributed Artificial Intelligence.

1.1 Context

The agent-based project arises from the necessity of real world military warfare simulation and validation, regarding the requisites of SAFEPORT's scope proposed by NATO. The main goal of this project is to validate a set of surveillance defense force strategies in a given mapped region, using the implementation of sensors, behaviors and communication process in a simulation

platform. The simulation only takes into account the maritime forces, despising the aerial and land means for harbor protection.

This project is framed in Multi-Agent System (MAS) scope, and has the main purpose of developing and agent-based architecture that fulfills the proposed simulation problem. Regarding this project context, the notion of agent is defined as a single real vehicle, with collective and/or individual communicative and interaction properties for the problem solving fulfillment. All the objects are intended to perceive information from its associated simulated environment, being an inherent property of the developed architecture [Woo95], providing a real world representation [Woo02].

The simulation property as temporal component should be implemented regarding the modeling of multi-agent system like strategies and behavior definition, being modeling a sub-domain of simulation. This project should also replicate the time conditions using the usual time step approach, creating a continuous simulation instead of discrete, like used in, e.g. state machines.

1.2 Motivation

Defense system represents a problem to be solved, firstly in personal context, then between groups and social entities, and finally between nations and countries, which entails in the validation of defense and surveillance strategies for the entities protection.

This project has a main motivation of developing a capable system which allows to increase the safety of valuable entities, regarding the absence of viable and easy real simulation process. Hence, the increasing simulation defense components and frameworks positively influence both sociologic and political levels, allowing a social stability in which investments and production are all in favor of population. One of the simulation purposes is to avoid costs and life-threatening situations that occur in real world training exercises, being a safe tool for simple decision-making processes. Nowadays the models used in simulation platforms cannot replicate with hundred per cent confidence the real world dynamics, being also commonly used for discarding situations that are not viable, but are expensive and dangerous in real world applications.

The usage of agent-based systems is also a major motivation for the platform development, since the military approach is yet closely attached to the differential equations for the modeling of non-linear complex systems. The usage of agents that represent independent and individual entities in the real world environment allows the creation of simple and different models for all the agents that constitutes the multi-agent architecture, along with ontologies and a protocol communication, resulting in a heterogeneous and complex system.

1.3 Problem and Goals

This simulation platform is framed in a higher project called SAFEPORT proposed by NATO, and is composed by different modules that should be integrated with. The platform should simulate the modeled dynamics taking into account the simulation characteristics, region mapping

and vehicle properties, provided by a configuration file from the DSS. The simulator should provide some feedback from the simulation execution, in which parameters like unseen threats in protected areas should be analyzed. Hence, the platform has to promote a bi-directional modular communication for the proper execution of the project as a whole, receiving the configuration and characteristics to simulate and provide report of simulation execution.

The main goal of the project is to replicate the real world dynamics regarding the military warfare, by using a multi-agent system. The simulation platform was only requested due to paradigm changing, which is intended to be a new type of simulation process in the military context, despising all the previous used methods. The final result should provide a viable and effective validation of the simulation execution, being capable of advising the user for the decision-making of defense configuration choice.

1.4 Document Structure

The structure of this document is based on different sections that are intended to organize the information, promoting an easy understanding of the dissertations purposes, and fast access to specific information which users want to be explained.

The first contents regards Section 2 and refers to the state of the art, in which will be presented some platforms used in the military modeling and simulation, giving an overview for the user's context in the scope of the dissertation. Section 3 is intended to provide a context of multi-agent system and agent's properties, and what is valuable to replicate for the real world warfare simulation. Section 5 talks about the implementation of the project, explaining all the details like simulation process and obstacle avoidance algorithms. Section 6 presents two case studies that are intended to validate the simulation's implementation, using real world cases to compare the final provided results. Section 7 is the final exposure of the dissertation, and promotes a discussion of all the methodologies used for the project implementation, the technologies used and the results for validation purposes.

Chapter 2

State of the Art

The scope of this chapter is to present some simulation and modeling works, providing a state of the art in the military context allowing users to understand how platform development is growing and the most important produced simulation software.

In this section will be also presented the most common problems that lead to the usage and development of this kind of platforms, as well as the necessary information for the military conception of simulation.

2.1 Introduction

In this section will be presented few modeling and simulation platforms within the military scope, making an analysis of how this kind of platforms can advise in the decision-making process of the best force configuration, complemented with behavior models of individual and collective entities. Also in this section will be analyzed some platforms which use agent-based architectures and class model that well represent the modeling and simulation process. One of the main goals is to create a comparison between the different analyzed architectures to be measured the pros and cons, for the further justification of framework's usage. The platforms to be analyzed were chosen taking into account its importance in the field, its implementation and its usage. These platforms go from early eighties to nowadays, creating this way a comparative term for the simulation platform's evolution. Regarding the Modeling and Simulation concept (M&S), a definition was conceived by the United States Department of Defense [oD98], which says:

"The use of models, including emulators, prototypes, simulators, and stimulators, either statically or over time, to develop data as a basis for making managerial or technical decisions."

The next sections refer to the independent treatment of Modeling and Simulation, for the specific explanation and good reader's understanding of concepts.

2.2 Simulation

Regarding the definition of simulation, a reliable source of how the terms should be used refers to the United States Department of Defense [oD98]:

"A method for implementing a model over time."

Other interesting definition is the appliance of the simulation process:

"The executing software on a host computer that models all or part of the representation of one or more simulation entities. The simulation application represents or 'simulates' real-world phenomena for the purpose of training, analysis, or experimentation."

In other words, the simulation is the attempt of real world replication allowing the possibility of behavior analysis, which entails in problem solving and aid in decision-making process for better behavior's applicability in the real environments. For a good understanding about the simulation concept, there is other concept that cannot be set aside and needs to be explained – Computational Modeling. The computational simulation concept involves the computational modeling due to the fact of modeled behaviors to execute, usually mathematical modeling for the creation of real world environment, needs a simulation process that can guarantee the well execution of the model overtime, being also possible the usage of the simulation process for further modeling cases. Figure 2.1 is intended to show a more realist goal of the simulation process, in which can be seen the simulation of a military system within a warfare environment. From the simulation process could emerge valuable behaviors that should be used for the problem solving improvement and system's calibration. An example of this approach is the Genetic Algorithms that use solutions of the problem for the generation of new ones, which entails in the further inclusion of new solution's generation.

As previously said, the main purpose of simulation implementation is related to the gathering of answers to a certain problem, using the real world replication. This replication should take into account all the environment dynamics like properties and behaviors, in which could be implemented in a computational system, avoiding the life-threatening and major costs of real simulation and training process.

Simulation is in an increasing importance process, due to the high level of abstraction and flexibility of the implementation diversity. Physics, economics, psychology, business logistics, military operations and sports are few examples of possible integration of simulation process for problem solving. All the real world environments which have a temporal component with a high degree of interaction could be computationally implemented to reach validation or problem solving purposes, using simulation processes.

Few real examples of simulation goals are related to optimization problems, environment control tests, group and individual training and decision support systems. One proper application of this type of platforms, is the simulation of complex systems which are impossible to solve analytically due to a high number of degrees of freedom inherent to environment dynamics.

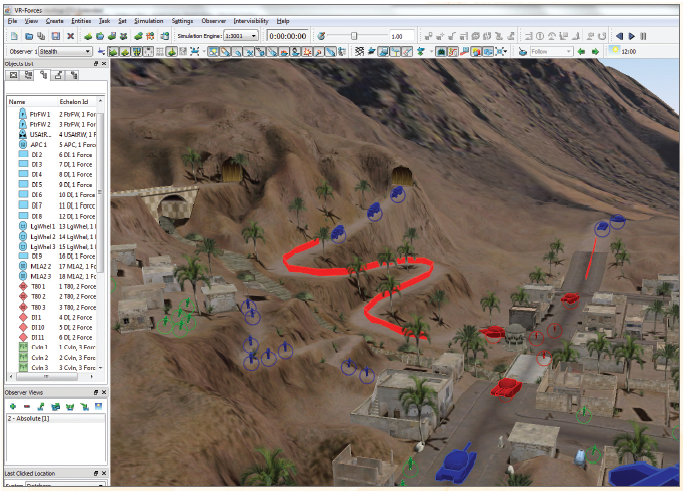

Figure 2.1: Military Simulation Environment [MAK11]

The execution of simulation environments which are computationally modeled could have different temporal complexities, going from few seconds – simple environment to model without the existence of a high number variables and degrees of freedom – to several hours – non-linear dynamic systems – using a web of computers, making usage of distributed processing.

As an example of real world replication, two projects that marked the simulation and modeling positively should be presented. The first one refers to the modeling of a complex protein producer in organisms using 2.64 millions of atoms [oPH05], and the second regards the modeling of 66.239 war tanks and vehicles, where the simulation environment was very similar to Kuwait, using multiple super computers of United States Department of Defense, in the scope of *High Performance Computer Modernization Program* [JPL97].

Like in all models created by humans, there is always an attempt of real world similarity, in which few of them were good, and others not. This similarity attempt emerges from the concern of both modeling entities and final users of the platforms. Hence, one of the most common problems of this context is to know if the final result of the modeling process is adequate, or if it fulfills the goals whereby it was created. For this problem solving, some authors like *Sargent* and *Smith* presented three essential phases: Verification, Validation and Accreditation [Sar91].

Robert G. Sargent presented in 1982 a simplified model of verification and validation that refers to the development process of models, in which authors like *Banks, Gerstein and Searles*, claims that this simplified model is more adequate regarding previous approaches.

All the presented phases are an integral part of the development simulation cycle and assumes firstly the world replication as a computational system, and then the conceptual model could be defined [CR07].

Figure 2.2 was build taking into account the *Sargent*'s diagram and could be seen in this simplified model, that are three different important sections. One of them is *Problem Entity* which is the system to be modeled; the other is the *Mathematical Model* that refers to conceptual model

construction; and finally the *Computerized Model* which is the mathematical model implementation as a platform.

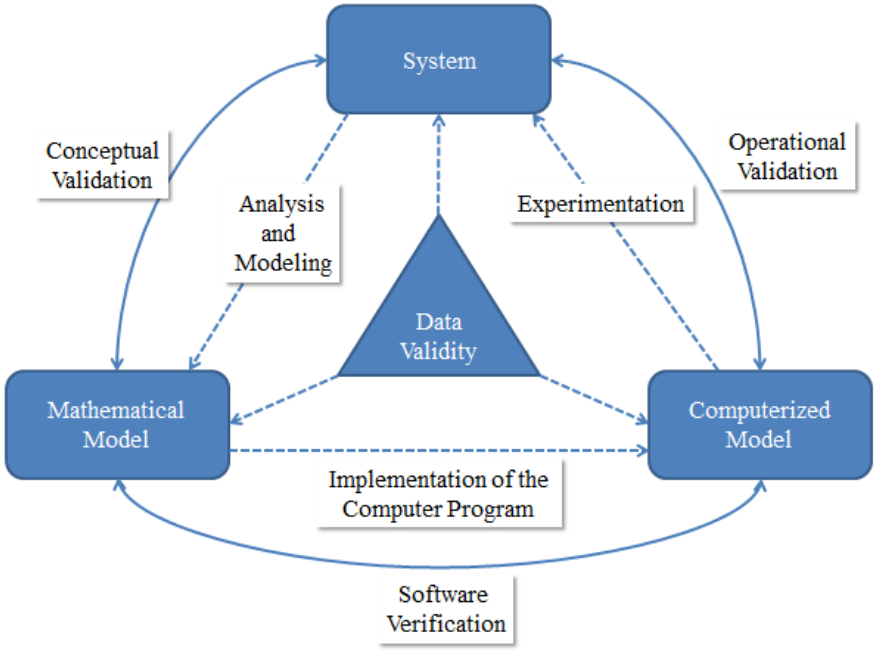

Figure 2.2: Verification, Validation and Accreditation Diagram [Sar91]

The conceptual model is built through the modeling and analysis phase; the computerized model from implementation of the computer program phase; and system is composed by the experimentation and operational validation phase.

Validation of conceptual model, or just Validation, is the process to determine if the gathered information and assumptions of the model are correct, and the system implementation provides the final expected results. This phase is commonly described as the answer to the following question: *Are we building the right product?*

Verification of the computerized model, or just Verification, is intended to ensure the correct computerized implementation of the conceptual model. This phase is usually described as the answer to the following question: *Are we building the product right?*.

Accreditation or Operational Validation consists of verifying if the final resultant behavior of the model is enough precise to its final purposes. The central section of Figure 2.2 – Data Validity – is responsible for ensuring that all the necessary data for developing, evaluation and model testing, and experimentation phase are adequate and correct.

Simulation Process

From the first attempts of simulation to nowadays, the modeling and simulation techniques

have been evolving in a way of creating the right and efficient process, which could validate the necessary parameters for the proper construction of simulation and model. This process goes from problem definition, software model construction, to its possible expansion for a better world representation. Figure 2.3 presents an adapted development process created by Smith in 1998, for the efficient and effective simulation's implementation, taking into account the validation and verification of the model.

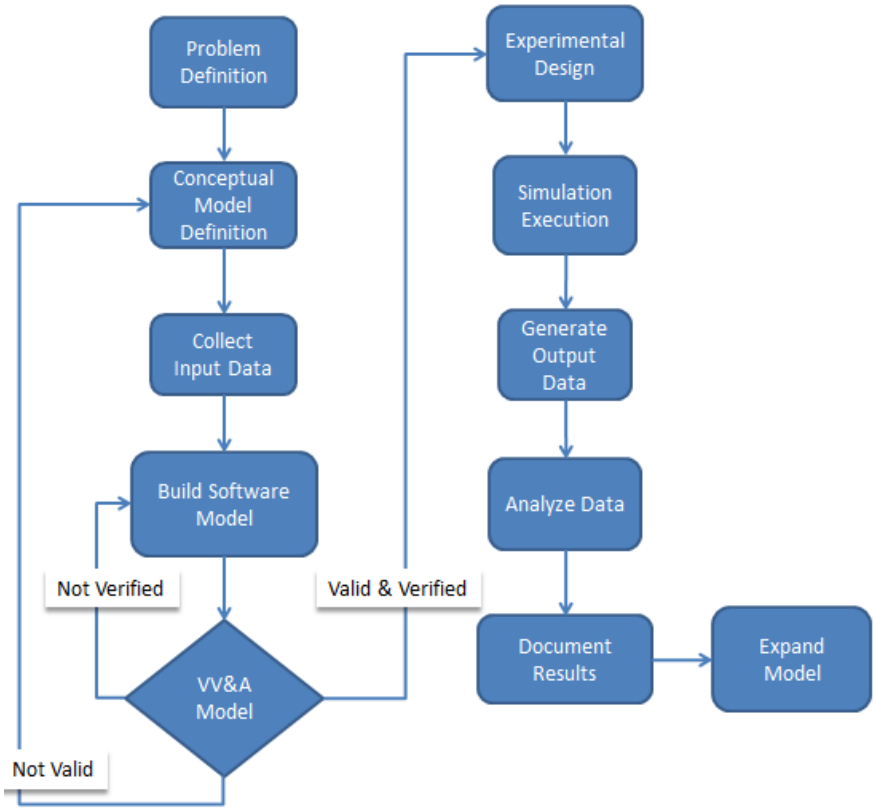

Figure 2.3: Simulation Development Process [CR07]

The first stage of this process regards the *Problem Definition*. This stage is intended to define the goals and requisites that should be taken into account for the system's development, and the accuracy inherent of the expected final results. The following stage is *Conceptual Model Definition*, which consists in algorithms usage for a possible modeling and its input and output. This stage should be repeated as necessary, in order to the proper model could be found. The next stage is constituted by the *Collect Input Data* that fits for parameters modeling purposes. This information should be also used for the validation of final results in which the proper correct behavior of the system can be tested with the means used to develop it. The actual stage represents the more complicated one, since it is related to human subjectivity for the notion of proper replication of the real world. *Build Software Model* is the next stage of the process, and it's based on the computational implementation, using a programming or specification language, of mathematical modeling and previous developed functions. However, according to *Verification & Validation*, all

the previous stages could be repeated, refining the model. The following stage it the *Experimental Design* which consists in the system design that can benefit the final result in terms of efficiency, reducing the time of simulation. *Simulation Execution* is the next stage of the process, and is intended to execute several simulations for the high number of results gathering. In the posterior stage, *Generate Output Data*, the Monte Carlo method is commonly used, in which different data could be collected, and this way the *Analyze Data* stage can be performed, being possible to infer with more efficiency the simulated behaviors and trends, that can answer the all the questions of the problem. The usage of tools that could represent well all this generated information is an asset for a quick answer attainment to the problem, as well as an easy explanation to other not specialized entities. The stage that succeeds all the aforementioned, is *Document Results*, in which all the process information is documented and registered for further analysis and sending to interested parties. The final stage of the process is *Expand Model*, and it is intended to expand the simulation model to other kind of simulation processes. By the simulation process development presented previously, different kinds of simulation could be implemented, being continuous, discrete or even stochastic.

Continuous Simulation

Those kind of simulations are likely used when exists the necessity of continuous variable modeling, representing an evolution and progression through time, regarding a certain environment. State variables that are associated to temporal processes are usually represented with differential equations, which are intended to reproduce the state of the environment along time. Continuous simulation is commonly used in contexts like economics, engineering, physics, biology, due to the usage of differential equations for the model construction to be used in simulation. The first times that a computer was used for the continuous simulation performance dates back to 1946, and in that time the necessity of solving diverse world problem was a real concern, promoting the computer functions like celestial mechanics, biologic systems and fluid mechanics. One of the most well-known cases refers to the predator-prey model presented by Lotka. This model consists in the interaction between two populations with the main goal of surviving, in which one of them is considered passive (prey), and only has the capability to eat, grow and reproduce, and the other (predator) depends on its preys to feed [Lot20] and [Vol26].

Discrete Simulation

Discrete simulation can be seen as a set of sequenced operations that represent the actual state of a dynamic system. The composition of the actual state is made through merging all the states of the system entities, and it is changed when events occur leading to the system update. Those changes are made instantaneously, and contrary to the continuous simulation, a transitory phase doesn't exist that can influence the environment, and the other agents indirectly. An example that

could be presented is the NDFA (Nondeterministic Finite Automata), which does not allow a continuous time line, but a discrete representation of the dynamic system, skipping from state to state. This kind of simulation uses concepts like queues, implementing most of the time data structures like FIFO (First In First Out) and LIFO (Last In First Out) for the organization of simulation state and preserve the good execution of the process [CR07].

Stochastic Simulation

The stochastic simulation is mostly used when exists a variation of certain parameter values that influences the state of dynamic systems along time [CR07]. This model is deterministic, changing the used simulation process, passing to be performed through statistic distributions, like Normal, Poison or Negative Exponential. To the model constitution, information about the variables' behavior is needed to calculate the proper model for simulation usage. This information is used to replicate a more reliable approach of the real world dynamics, promoting a good simulation execution and more adequate results.

2.2.1 Military Context

For modeling cases of military environments, *Andrew Ilachinski* claims that the hostile environment in which military means are subjected could be modeled as an adaptive complex system. The hostile environment between entities could be mathematically and physically modeled, through a non-linear dynamic system composed by the interaction of hierarchical organized agents which continuously adapt to the environment [Ila04].

Regarding the modeling environments like nonlinear dynamic systems, there are different kinds of environments, with distinct characteristics. One of those environments is intended to locally control its operations, in which the entity group is disorganized and has not the adaptive skills. Other situation refers to warfare environments that hardly can reach communication equilibrium, and act directly through the environment. Finally, can exist an environment in which its agents have an high adaptive property, knowing that sometimes a regulator agent couldn't exist to coordinate and organize its behavior, leading to a survival environment. For an easy understanding in military operations, *Red Team* and *Blue Team* are the names of strike and defense teams, correspondently, in which an hostile environment should be modeled.

2.2.2 Simulation Platforms

Despising the platform's explanation in 2.2.3, this section is intended to discuss the platforms that are not agent-based giving an much wider overview about the military simulation platforms to users. Will be presented the simulation and validation process of none agent-based platforms, in which the user must provide the strategy to the software of its environment applications. Section 2.2.3 will be focusing the explanation of platforms which provides means to the behavior modeling through adaptive functions and implementations.

Network Centric Forces

These platforms consist in simulating the conflict and communication processes in warfare environments, and represents one of the major projects developed nowadays regarding in one hand the easy way of simulation parameterization, and for another the realistic representation of the simulation process. This platform is constituted by other two platforms, one of them is named VR—FORCES, developed by VT MAK, and the name of the other is QualNet, developed by Scalable Network Technologies. Both of platforms have different purposes, constituting essential parts of communication, simulation and interaction with users.

VR-FORCES platform has the major goal to simulate aerial, land and sea entities' behavior, using different types of assets, in which users can previously model the behavior and strategy of vehicles and forces that should be implemented in certain situations and contexts. This way, the platform has a three dimensional graphical interface where conflicts should be simulated and presented for the user to follow the evolution of its strategy's implementation.

The QualNet platform is a realistic communication simulation between agents and entities that constitutes the simulation's configuration. In a hostile situation, some limitations could exist, like physic limitations, which can reduce or avoid the direct communication between entities. Those limitations could be provoked by land characteristics, like urban environment, or the range of communications, like establishing a message exchange with a satellite. The communication simulation goes from the human interaction, like soldier level communication, to communication between antennas placed in different buildings in a city.

2.2.3 Agent Based

This subsection is intended to show few agent-based simulation platforms that are intended to complement a modeling case of agent behavior, and agent simulation with no generation of new behavior through agents' adaption.

Rand Strategy Assessment Center

The RSAC software system was developed to improve the strategy analysis, combining the best properties of War Gaming, and analytic modeling, using a multi-agent architecture based in rules that describes its behaviors [Hal85]. Those rules are defined by people with military knowledge, in a most alike English language called ABEL, which can offer a human independence, allowing the automatic simulation of agent behavior. The war gaming concept is itself a discipline with a certain degree of realism, but doesn't allow the multi-scenario analysis, meaning that is not possible to analyze the simulation entities' behavior, inferring this way if a certain strategy is effective or not.

The platform is constituted by four different types of agents, which are intended to simulate the Red Team, the Blue Team, the environment of the simulation process (Scenario Agent) and the coordinator agent (Force Agent). The behavior of the first three agents is written in ABEL,

and the fourth is written in C programming language. The first two agents, Red and Blue team, aims in simulating the behavior of two conflict forces, while the scenario agent has the objective of simulate the effects of environment dynamics. The coordinator agent is intended to combine and organize the effects of those three previously explained agents.

The implementation of both Red and Blue teams constituted by a structure divided into three distinct levels – National Command Level (NCL), Area Command Level (ACL) and Tactical Command Level/ForceAgent (TCL) – representing an hierarchical decision-making. NCL receives influence from the actual environment state to determine:

- Context of the decision-making to be performed of each team;
- The operational goals that each agent should reach;
- Operational strategy that each agent should implement for the goals fulfillment.

Determining the above parameters, the NCL has the capability of choose the Analytic War Plan (AWP), constituted by a set of rules with the main goal of continuing the warfare simulation. This platform has previously calculated all the possible plans that could be implemented during the conflict phase.

After that, the previously calculated plan is sent to ACL with the intent of being implemented, sending information for the scenario and force agents. Finally, the TCL, which is merged with the coordinator agent, aims in simulate the execution along time.

One peculiar characteristic of this platform is instead of the a direct communication between agents, the state of each entity of the simulation is changed and updated through the World Situation Dataset (WSDS). For the indirect managing of contents, a dictionary is used allowing to know the information that should be given to an agent, when it is requested.

ISAAC

The name ISAAC comes from *Irreducible Semi-autonomous Adaptive Combat* and is an agent-based platform for the simulation of a reduced number of combat forces. For the specific context of this platform, a multi-agent simulation is artificial life community capable of develop an adaptive complex system [Ila04]. In the early years, this system allowed to demonstrate that the usage of agents embedded in a military simulation is possible and viable. This concept was presented in 1997 and is agent-based, in which follow simple rules of behavior that could be associated to a real world situation with military properties. The development environment of this platform was MicroSoft Disk Operating System (MS-DOS), using C programming language.

ISAAC – Agent Architecture

As previously said, the agent concept is the base of this platform development, and this way each of them has some properties that define it as an independent entity, and behaves in order

to reach its goals. Each agent represents a unit of warfare, in which could be a single soldier, a transportation vehicle, an armored vehicle, etc. to guarantee the agent's operability were defined the following characateristics [Ila04]:

- Doctrine: a default local-rule set specifying how to act in a generic environment;
- Mission: goals directing behavior;
- Situational Awareness: sensors generating an internal map of environment;
- Adaptability: an internal mechanism to alter behavior and/or rules.

All agents with those characteristics comply with a hierarchy that is vertically defined. This verticality is due to the fact of a commandant existence which could locally influence the agents, to a group or individual, or globally to all the agents in the environment. These agents have the main goal to simulate directives of military situation, in which high senior posts have different goals and behavior from its decendents.

EINSTein

The name EINSTein derived from *Enhanced ISAAC Neural Simulation Toolkit*, and is an agent-based simulation platform like ISAAC. This system uses the concept of agent to allow the autonomous individual and group behavior development, for further integration with the simulation environment. This platform uses the same philosophy of ISAAC platform, which an agent is an individual representation of people, vehicles, or generally, entities discarding the notion of agent like weapons, trying to replicate a machine's behavior.

The EINSTein platform can be considered as the first attempt to simulate the warfare using agent-based architecture, from small to medium scale [Ila04]. Being the agent's implementation a simple and rudimentary intelligence approach, they are prepared to react to several situations with a large mechanized ways, which can depend from the warfare environment dynamics. The existence of dynamic and adaptive rules allows the agent to respond rapidly to unexpected situations providing the best response, knowing which behavior should be applying in certain circumstances. The platform was conceived from object oriented architecture, creating a basic representation based on objects. This type of programming is mostly associated with a class model that represents the relations and interactions of objects. Class model allows the specification of an adequate architecture for the agent-based implementation being associated to concepts like heritage, polymorphism and encapsulation. Despising the implementation of the agent-based architecture, this platform is divided into three independent modules:

- Combat Engine;
- GUI;
- Data Collection.

This type of approach allows those three components being compiled independently, without the necessity of the whole platform's compilation that does not represent any considerable change. Since this platform is divided into three independent modules, a file based communication was adopted for the modular information exchange. The communication is based on information request to promote the actions and some complementary information due to the well function of the system. A basic example is GUI information request to the Combat Engine to update the users information, maintaining this way the normal workflow of the simulation process as a whole taking into account all the modules.

EINSTein - Behaviors

Other strong component of this platform is the set of behaviors based in military actions that can fully replicate the real world warfare, promoting greatly the simulation's final result. A subset of behaviors could be represented as following:

- Forward advance;
- Frontal attack;
- Local clustering;
- Penetration;
- Feints;
- Retreat;
- Attack posturing;
- Containment;
- Flanking maneuvers;
- "Guerrilla-like" assaults.

One of the platform's goals is to generate and obtain a new set of behaviors and its subsequence evaluation in terms of macro-behavior – general behavior of teams – and micro-behaviors – individual behavior within a team. These behaviors patterns are used as new strategies to move and act in hostile environments, going from the operation coordination to the individual survival behavior. This platform can be seen as a higher goal regarding the purpose of its creation. If the main concept of this system could be abstracted and be applied in any simulation environment, the changes in micro-behaviors could be explored by knowing the impact in macro-behaviors. This type of approach contributes to the *Chaos Theory* understanding, being useful for complex system development through slight changes in agents' behaviors, promoting the problem solving effectiveness.

2.3 Modeling

One of the most well-known Modeling definition was made by United States Department of Defense:

"Application of a standard, rigorous, structured methodology to create and validate a physical, mathematical, or otherwise logical representation of a system, entity, phenomenon, or process."

In other words, modeling could be seen as the implementation of several models, being model a real world reproduction in a computational application. The usage of models in simulation processes is a necessity due to its behavior execution in a certain environment, also modeled, where different behaviors should interact resulting in a similar real world replication. The following section discusses some important notions of modeling understanding, regarding its usage and construction, by presenting examples of military simulation and authors' theories using computational modeling.

Lanchester's Law

For a good understanding of military simulation in terms of mathematical approaches, regarding the last century, a travel to 1914 should be made to know the presented work of *F. W. Lanchester*. This author's legacy starts the military modeling and simulation history in warfare context, being yet nowadays used. Despite only few mathematical work be presented in this section, authors like Lottka-Voltera, Chase and Osipov should be noticed and taken as a reference in this document, being the precursors of combat theories and predator-prey model.

Lanchester's Law - Equations

One of the most simple cases that *Lanchester* presented as to do with the directed fire or Square Law [Lan56]. The author claims the attriction of force (Red or Blue team) is proportional to the number of elements of the opponent force. Assuming *R(t)* and *B(t)* are a quantitative representation of Red and Blue teams' strength, in a given period of time *t*, and α_R and α_B are a representation of constant effective fire rates at which one unit of strength on one side causes attrition of the other side's forces. Hence, *Lanchester* built and directed fire model as can be seen in Figure 2.4.

$$\begin{cases} \frac{dR}{dt} = -\alpha_B B(t), R(0) = R_0 \\ \frac{dB}{dt} = -\alpha_R R(t), B(0) = B_0 \end{cases}$$

Figure 2.4: Directed fire model equation

Figure 2.5 presents an approximated way of solving differential equations, using hyperbolic functions.

$$\begin{cases} R(t) = R_0 cosh(t\sqrt{\alpha_B \alpha_R} - B_0\sqrt{\frac{\alpha_B}{\alpha_R}} sinh(t\sqrt{\alpha_B \alpha_R})) \\ B(t) = B_0 cosh(t\sqrt{\alpha_B \alpha_R} - R_0\sqrt{\frac{\alpha_B}{\alpha_R}} sinh(t\sqrt{\alpha_B \alpha_R})) \end{cases}$$

Figure 2.5: Hiperbolic Function Definition

The model presented in Figure 2.4 is resolved with equation presented in 2.6.

$$\alpha_R(R_0^2 - R(t)^2) = \alpha_B(B_0^2 - B(t)^2)$$

Figure 2.6: Directed Fire model solved

As *Ilachinski* claims, despite of the simplicity of presented equations, is hard to find enough direct and truth relation that validates the real world dynamics. The inconsistent data could be a detrimental factor for the model replication to a real warfare situation, much due to the information that only consider one team, not having the proper information about both. Other case is the uncertainty of how information is documented not having the proper formality, reaching data like *killed*, *killed* + *wounded*, *killed* + *missing*, etc [Ila04].

Lanchester's Law - Limitations

As previously said, this mathematical model cannot describe a sufficient approximation of reality. One of the strong limitations is the outdated information of new war strategies and used technology in twentieth century, which means well trained autonomous teams, with new armor that can be inserted in a cooperative environment, that is not contemplated in Lanchester's Law [Ila04].

Chapter 3

Simulation

3.1 Multi-agent Simulation

3.1.1 Multi-agent System

Multi-agent systems can be defined as a set of agents that interact in a common environment, having the ability of changing themselves and the environment [Fer96]. A multi-agent system can be also seen as a society, which is a set of coexisting independent entities, reaching its goals, using its cooperation and communication skills, like interaction, perception, adaptation and mobility. This type of system has the capability of solving its own problems through interaction to achieve its objectives [Oli99]. Another definition of multi-agent system involves a flexible web of responsible entities aiming the problem resolution, working as a group to obtain a collective answer which is beyond the individual knowledge of each entity [DC89]. Figure 1 represents some important characteristics like an interactive environment where entities are inserted in, indirect communication through each visibility and influence circle, and direct communication by the organizational relationship and interaction links. These communications are a good representation of real world where entities are neither omnipresent, neither omniscient.

For the correct implementation of a Multi-agent System, the following six topics should be taken into account [Fer95]:

- An environment that is usually a space;
- A set of objects that in a given moment can associate any object with a position;
- An assembly of agents, which are specific objects that represent the active entities in the system;
- An assembly of relations that link objects to one another;
- An assembly of operations allowing the agents to perceive, produce, transform, and manipulate;

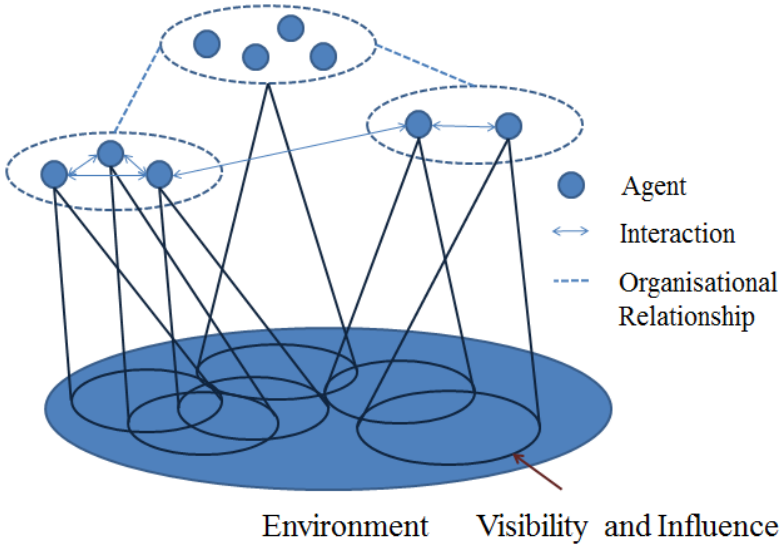

Figure 3.1: Multi-agent System

- Operators with the task of representing the application of these operations and the reaction to the world.

Those characteristics allow the correct classification and segmentation of agent-based system, in which its construction should be formalized to direct the problem resolution through the architecture conception phase. For the development of a multi-agent system there are some pertinent questions, regarding its context areas, that should be raised to promote a correct coexistence between objects that constitutes the environment [Bou04]:

- Decision-making: Which decision-making mechanisms are available to agents? How agent perception, representation and act is modeled?
- Control: Are there some hierarchy between agents?
- Communication: Synchronous acting of agents? Which type of agent messages should be considered? Is there a specific syntax?

It is known that the architecture and agents must be specific for each presented problem; hence these topics constitute the cornerstones of MAS that should provide an approximation of real world dynamics, and have to be implemented and modeled in order to reach a valid solution.

3.1.2 Agent

The definition of agent varies in ranges of context, assuming different functions and purposes in areas like philosophy, sociology, economy, law, and others. Despite those contexts, the definition is related to Artificial Intelligence: An autonomous agent is a system situated within and a part of

an environment that senses the environment and acts on it, over time, in pursuit of its own agenda and so as to affect what it senses in the future [Fra97]. Other definition is: A computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its design objective [Woo02]. Agents as software implementation follow certain properties that define and allow the fulfillment of its purposes. There is a set of characteristics that models an agent inserted in an environment that allows an approximation of real world properties [Woo95]. The characteristics that model a simple agent, also known as weak notion, are:

- **Autonomy**: The agent operates without the direct intervention of humans or others;
- **Social ability**: Interaction with other agents;
- **Reactivity**: Perception and reaction of environment changes;
- **Pro-activeness**: Taking self-initiative;
- **Temporal continuity**: The agent is continuously running processes.

More complex characteristics that an agent should have to resemble with humans are defined by the following:

- **Rationality**: Act in order to achieve its own goals;
- **Adaptivity**: Adjust itself to the habits, working methods and preferences;
- **Benevolence**: An agent will always try to do what it is asked for;
- **Collaboration**: An agent should not unthinkingly accept certain orders that could put in danger the environment or damage other agents;
- **Mobility**: The ability to move around the environment.

For the environment to implement the minimum complexity, an agent couldn't have the whole perception of it, being this way omnipresent, and have control of the environment if it has the capability to change and influence it, provided by the interaction properties. Hence, the agent is a computational system intended to simulate the behavior of an entity to achieve its own goals, interacting with the environment and other agents.

3.1.3 Environment

The environments in which agents are inserted and emerged have different kind of properties that determine its behaviors, collectively and individually, and each one should execute the corresponding actions along time to reach goals fulfillment. Those properties could be described as [Woo02]:

- *Accessible versus Inaccessible*: The accessible environment is characterized by providing to agents, when required, full, precise and updated information about it. Most of the realistic environments are not accessible because information cannot be available due to environment or entity constraints;

- *Deterministic versus Non-deterministic*: A deterministic environment is characterized for the production of same result for the certain action, not existing uncertainty in its execution. The realistic environments are all non-deterministic due to complex system properties that cannot be securely predicted;

- *Static versus Dynamic*: A static environment could be described as non-changeable, despite when agent performs some action. In other hand, dynamic environments vary independently from agent's action, being the most alike with the real world environment;

- *Discrete versus Continuous*: An environment could be defined as discrete, if it has a fixed, finite and limited number of actions, like a DFA in which has a set of states that can describe the actual environment, not existing a temporal component for the action's execution.

Regarding the accessibility characteristic, *Wooldridge* claims that as much an environment is accessible, much easier would be the construction process of an agent, being more efficient its behavior modeling. This idea is easily justifiable since an agent is an independent entity that always makes the correct choice, being an accessible environment a good factor for its correct implementation.

The deterministic environments are preferable regarding the agent's construction, due to the uncertainty associated to agent's action. In other words, an action that has always the same result doesn't have to measure the repercussions, in the environment and in other agents, of its actions, trying to deal with exceptions that can occur.

The construction of agents, with cooperative or independent properties, is hampered by the ineffective capability of plan construction, regarding a dynamic environment. The production of plans is very used in the context of MAS, being very effective in static environments, in which the process of plan conception takes into account the non-changeable properties that can guarantee always the expected execution of actions. Being a plan a set of tasks for further execution, in a context that environment changing is certain, the prediction of tasks effects is not viable [AT90].

The chess game is by itself a discrete environment, in which the number of board configurations is very high but finite, making an agent implementation in chess context more efficient comparing with continuous environment. One of the difficulties regarding the continuous environment is associated to computer themselves as a discrete space and not continuous. The simulation conditions could be performed, but will always exist an incompatibility between the finite physical resources from computer and the continuous environment to simulate. Some information has to be lost in the process for the simulation to continue, lowering this way the precision of agent's behavior [Woo02].

3.2 Military Simulation

In military simulation, there are different accuracy levels and real world representations that goes from *Major Field Exercises* to *Analytical Models*. The most alike real world simulations are the Major or Minor Field Exercises, in which the simulation executes in the field, taking into account the environment dynamics. This approach can guarantee a more reliable and accurate simulation in its *Operational Realism*, which entails in an increased cost. In other hand, *Analytical Models* are a totally different approach comparing with the real world implementation. Those models are intended to create an abstraction of the environment representation, in which behaviors and strategies could be computationally simulated, entailing in more *Convenience and Accessibility*. The purpose of both spectrum extremes are equal, being the validation of new behaviors for the real world implementation. Figure 3.2 shows the most used definitions for the military simulation implementation.

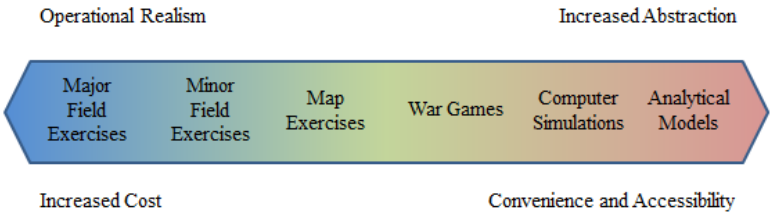

Figure 3.2: Military Simulation Range [Tay83]

3.2.1 Properties

For the proposed simulation platform, several properties should be considered for a minimum accuracy and reliable system's implementation. Some of those requirements are exposed in the underlying section, describing and defining the most important properties like obstacle avoidance, communication and speed.

- **Obstacle Avoidance**: The obstacle avoidance property regarding military warfare replication is one of the most important issues. Since certain vehicles could not undergo through certain regions due to its characteristics - like maritime assets that can only move in water conditions – the planning route between to different points is necessary. Along with this property, the shortest path could be also calculated for a more accurate replication;

- **Communication**: The communication process represents the interaction of different vehicles in the simulation. This property is intended to define the ontologies for the messaging understanding between entities, providing pertinent information for coordination and cooperation purposes;

- **Speed**: The capacity of vehicle's displacement through the environment regarding its characteristics is an important approach due to entities dynamics. As it is known, a defense configuration should be heterogeneous regarding the purposes of each vehicle, being this way necessary the replication of speed;
- **Sensors**: The environment perception is a characteristic inherent to the sensors of each vehicle. The sensors have the capability to perceive information of the environment, allowing the entity to act according to its own beliefs and external conditions. As said before, the characteristics of each entity are different, being necessary the flexibility in perception definition, e.g., perception radius. The sensor replication can be implemented using different types like *Navigation Radar*, *FLIR Camera* and *Daylight Camera*;
- **Effectors**: Effectors are intended to avoid the action's continuity of opposite force. It can be performed using several objects like *Long Range Acoustic Device*, *Optical Disruptor* and *Spot Light*. Most of these effectors' are performed in presence of enemy forces in protected areas, in which they should be captured, or interrupted.

Chapter 4

Problem and Motivation

Protection of cooperating nations like it's proposed by NATO is the main issue that is proposed to be solved. The competition for resources and force demonstration represents the new paradigm of war. Those new parameters of war are intended to play a more strategic and complex approaches, in which brute force and old strategies reveals not to be that effective and efficient nowadays.

The purpose of the platform's development arise from the difficulty of telling if a certain configuration is enough secure to promote an effective surveillance regarding a determined protected area. The problem has the main issue of selecting the best configuration that should be used for the harbor protection and surveillance.

4.1 Motivation (DAT-POW and Safeport)

The development and real warfare application of strategies and individual behaviors is hampered by life-threatening and multiple situations that should be validated. Simulation is a common process used in this type of circumstances due to the real life modeling along with all entities that make part of it. Latency of events, waiting time between strategies development and its testing in real life, entails in longer waiting times, which is usually an important parameter in the simulation process. The usage of a DSS along with a simulation environment is the approach used for the simulation of surveillance systems, producing a result which is a quality measure of the configurations being tested.

SAFEPORT is a project that aims to develop an application that will help the selection of best surveillance systems configuration for harbor and force protection. One of the components of the DSS will simulate the behavior of surveillance system and possible threats. These elements span from human forces units, to sensors, unmanned systems, and vessels. The simulation will use several configurations of the defensive elements (proposed by the optimization algorithms) against different kinds of attacks under various environmental conditions in a specific operational scenario. These simulations will help to enhance the proposed solutions and, at the same time, feed the DSS with the results of the several solutions targeting different metrics.

The final result of the SAFEPORT project will be made available to the NATO partners for planning the best defense configurations to force protection in a port or harbor environment.

4.2 Goals

For the implementation of this project the main goal is to simulate a given configuration of objects, which could be vehicles, sensors and a mapped region, along with associated behaviors and actions to each object within a common environment.

The usage of agent-based architecture to the simulation implementation is also an objective to reach in this project. One of the main dissertation's purpose is the validation of multi-agent systems implementation for the military simulation. The definition of different agent's architecture with heterogeneous behaviors and the communication protocol is intended to provide a warfare representation.

An interface is an important implementation in order to validate the behaviors and mapped regions within the simulation. The interface is supposed to allow the input of configuration files, as well as running the simulation with the common usability of it, like play, pause and stop bottom.

Finally, the main goal of this platform is to provide a reliable feedback to the DSS that should influence new posterior executions of the application as a whole. This feedback is intended to calibrate the DSS for further results be adequate for a given configuration file, being this way adaptive and sensitive to the simulation validation.

Chapter 5

Implementation

5.1 Platform

5.1.1 JADE - Multi-agent System Framework

The multi-agent system framework chosen for the implementation of the simulation project was JADE, which is a software framework that uses the Java programming language for the multi-agent system's implementation [Bel07]. JADE is FIPA-compliant – a setting computer software standard for agent development - meaning that implementations in the same or different programming language can interact within the same environment, creating a flexible approach for open systems that wishes the agents inlet and outlet. A FIPA standard also allows the proper communication between agents by defining ontologies for agent's understanding of exchanged messages within its associated container's.

Architecture

Regarding the creation of agents, it is always needed at least one identifier which can be the FIPA Agent Identifier (AID), being an agent label which distinguishes it unambiguously. Hence, an agent should be registered in a transport address in order to be identified for the multi-agent system's interaction. The execution of agents is made recurring to containers that host the agents that should communicate and interact in a multi-agent architecture's definition. In JADE, there's a principal container - *main-container* - that has a mandatory execution independent from the creation of others, and two different agents are created with it: the Directory Facilitator (DF) and Agent Management System (AMS). The first one is not mandatory, and provides yellow pages services - accurate, complete and timely list of agents – to all the authorized pages, and the second provides coverage of container's operations such as agent migration, creation and deletion. The JADE architecture allows the creation of several containers that could be distributed over the network, enabling the distribution of computer processing and the execution of different containers at the same time, in different locations. Figure 5.1 represents the relationship between the elements that composes JADE platform and the tools to promote a container's interaction.

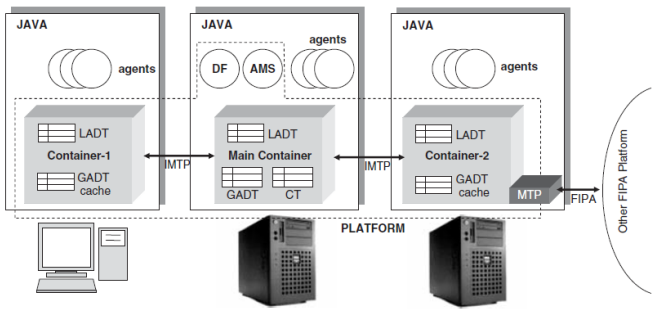

Figure 5.1: JADE architecture [Bel07]

The communication through agents living in different containers is possible using the Internal Message Transport Protocol (IMTP) and it is not mandatory the usage of FIPA standards for the interpretation of messages, since it is used for the internal platform communication only [Bel07]. For the identification of registered agents in the containers, a Global Agent Descriptor Table (GADT) is maintained in cache that provides its current status and location in the JADE's system.

The agent's execution is promoted by the usage of behaviors, which are tasks that should perform within an environment, with or without agent's interaction. Each agent can have several behaviors running at the same time, performing different tasks like, e.g. receiving and interpreting of others agent's messages, and explore the environment for mapping information. All the behaviors associated to an agent should implement an action and *done* function, for defining the way of acting and identifying if the behavior is or over, correspondently. The *action* method is called when the behavior should be performed, and *done* method is called when the behavior is terminated.

There are several types of behaviors that promote and facilitate the flexibility that an agent should have within an environment. As example of that are: one-shot behaviors – which perform the action method one time; cyclic behaviors – the action method is executed whenever it is called; generic behaviors – have associated a trigger to execute the action method.

5.1.2 FIPA - Foundation for Intelligent Physical Agents

Since JADE framework for agent modeling is used, the communication is based on FIPA-ACL Messages, which is one of the most used language [Koe04]. The ACL message is a simple ASCII codification, which increases the time processing due to the need of parsing and interpretation/-codification of results. The structure specification of ACL message was defined by Foundation for Intelligent Physical Agents [fIPA00] as a set of elements: Type of Communicative Act; Participants in Communication; Content of Message; Description of Content; and Control Conversation. The only mandatory element in ACL messages is the *performative* (Type of Communicative Act), but is expected to also be defined the *sender*, *receiver*, and *content*.

For the communication's understanding between agents, FIPA-ACL standard defined a set of communicative act – label that describes the intent of communication – which is the central guidance in the messaging interpretation when the communication protocols are implemented. The Library Specification of communicate acts is divided into different purposes like *interrogatives* – information query; *exercitives* – asks for an action to be performed; *referentials* – assertions about the world's environment; *phatics* – establish, prolong or interrupt communication; *paralinguistics* – relation between messages; *expressives* – attitudes, intentions or beliefs [Bel07].

5.2 Conceptual Solution

The description of project in overall terms gives the reader an general overview of its purpose, preparing for the best understanding of all the specific details and the justification of certain approaches. This section is the central explanation of the whole dissertation project, in which will be explained the physical architecture, logical architecture, class model, multi-agent architecture and agent architecture. Those subsections will be explained in depth for the user's full understanding of project along with the reasons that lead to its implementations.

For the model construction, several steps should be taken, like simulation, until the final model emerges [Per10]. The military modeling strategies could go through some phases of the model generation, but most of the times it is implemented in real-world, which is the approach that ensures more reliable results due to the huge existing gap between computational modeling and real-world dynamics. Hence, the real-world validation of military strategies is very expensive regarding its implementation using the assets that composes the simulation. Most of the times there are huge costs knowing if certain plans and tasks are effective in a warfare situation, but more important than that, since life-threatening could be a possibility, computational implementations for simulation gains total sense. Regarding these constraints, for the proper validation of military situations, several simulations should be done due to environment dynamics and the limitless exceptions that have to be analyzed and contemplated in the reformulation of the final model.

5.2.1 Physical Architecture

The physical architecture intends to show the interactions between outer components of the whole SAFEPORT project and the simulation platform, which is represented in Figure 5.2. For the underlying diagram, there will be explain in depth each interaction and its meaning within the project context.

As said before, the DSS should provide a configuration file with specified characteristics of the vehicles, sensors and mapped area are, being the source of the simulation. The details of proposed XML-based [Bra08] file is exposed in 5.2.3.

For the reading of the configuration file an user interface was created allowing users an easy and quick access to the means for the simulation start. It is in Java Simulator where all the processing is executed with the modeled multi-agent architecture and the definition of each agent. For the behavior modeling is used external information, allowing the flexibility for the users to change

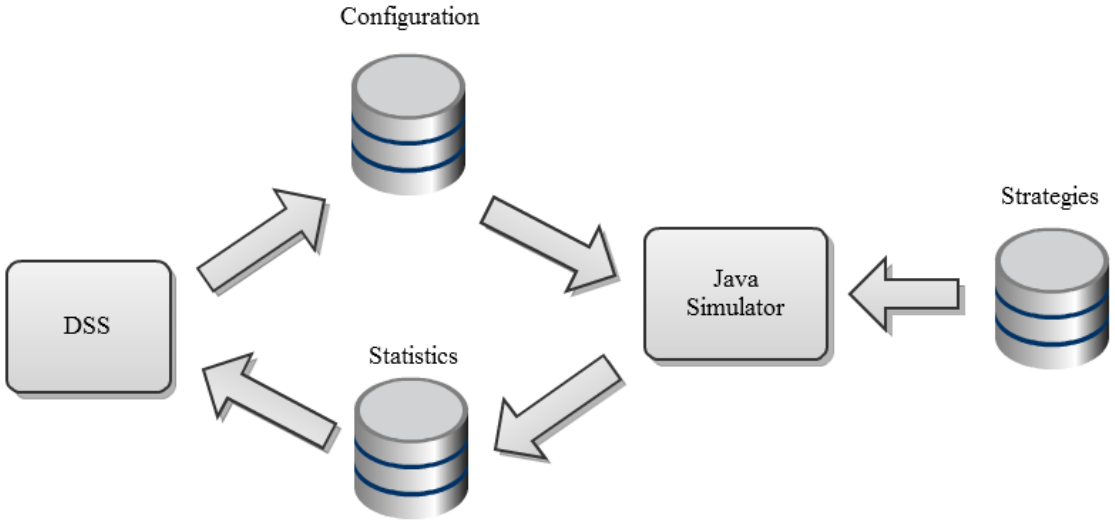

Figure 5.2: Physical Architecture

the behavior from simulation to simulation. These strategy files are explained in depth in 5.2.3. The configuration and strategy files are the external factors that should be provided for the proper simulation execution. When the simulation ends, a validation file is created with all the statistics of the simulation that contains all the information about the given configuration validation, for the DSS to analyze. All the workflow explanation should be used as a cycle until the best result attainment. As simulation don't reproduce the exact real-world dynamics, the usage of several simulations runs is a normal practice forming a cycle between the DSS and the Java Simulator

5.2.2 Logical Architecture

As it is known, for the development of a multifunction application it's necessary the usage of different technologies to reach the best solution. Hence, the logical architecture diagram, presented in Figure 5.3, is a representation of all the platform components that implement all the developed functionalities, from interface to multi-agent system.

Regarding the interface, it is composed by java SWING toolkit [RV03], for the user interaction, and Processing programming language [Rea07] for simulation animation. SWING is a Java GUI (Graphical User Interface) widget toolkit that is intended to provide visual components that allow the interaction between the user and platform, being Processing an open source programming language that could be developed in its IDE (Integrated Development Environment) or integrated with other languages like Java and C++, and different IDE's – e.g. Eclipse.

Those two different approaches are the core of the interface that provide, in one hand, an easy user interaction, and for another, a real-time simulation representation for behavior validation and observation. Is through SWING interaction that the simulation starts, using the control panel that allows start/continue, pause and stop the simulation. When the configuration file is inserted into

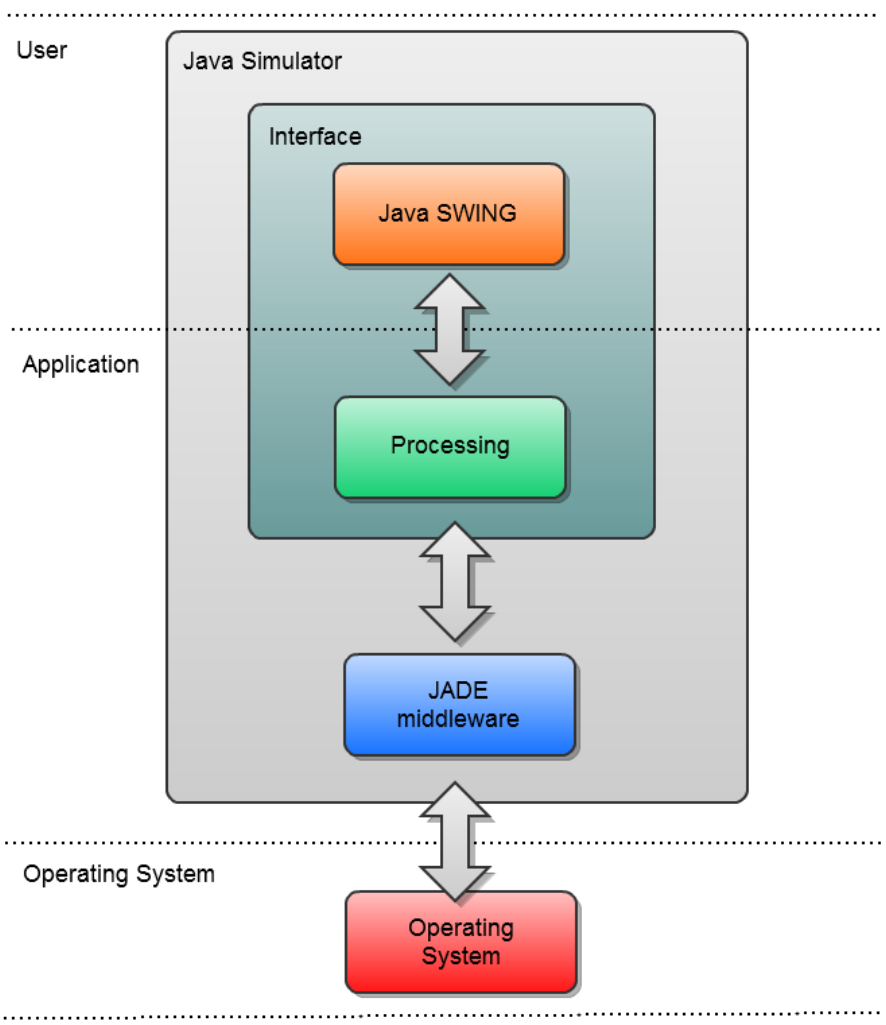

Figure 5.3: Logical Architecture

the platform, the corresponding number of agents and mapped region are created with the specified characteristics. For each agent could be associated a strategy, that is independently modeled by a XML-based file, giving user the possibility to adapt and update the simulation to be more adequate to its purposes of simulation. The interaction between agents and the multi-agent system only starts with the user's will.

Since the simulation is based on a multi-agent system, it should use an agent development framework for the multi-agent and agent architecture definition. To accomplish that, the JADE framework [Bel07] was used that includes a useful communication protocol – FIPA (Foundation for Intelligent Physical Agents). The whole multi-agent architecture was modeled in JADE along with each agent architecture. Both of implementations are based on previous studied architectures, but were adapted for the context of the problem. Being the simulation a mean to get the final result, it takes into account the beginnings of the components architecture implementation, using

the property of continuous time, represented by a single step, for the warfare execution. For the simulation an agent was created being all the execution module up to the JADE middleware. The execution of platform is represented by the Java Simulator which comprises the Java SWING, Processing and JADE middleware. Hence, all the external communication that a platform should logically have, is with the operating system, through the Java Virtual Machine (JVM), and the user, being all the remaining factors not included for the platform to interact.

One widely used platform is Repast [Nor06] which is an open-source modeling and simulation toolkit. Despite the whole simulation and modeling process, this platform has also an interface for the user's visualization of environment dynamics. The strong reason that led to the JADE platform choice was the more appropriate functionalities that JADE provide for the military warfare simulation comparing with Repast, being a simulation framework, and JADE a fully platform guided to the development of MAS. As said before, the interface is not a requisite of the system, being used for validation purposes, a point in favor of its usage. The main purpose of the simulator's platform is to replicate real-world warfare, using message exchanging, ontologies and a multi-agent architecture that implements the world dynamics, which is provided by the JADE framework in an easy and fast way.

5.2.3 Modular Communication

Configuration File

For the simulation platform some information is needed regarding in what conditions it should be started, executed and finished. Hence, a XML-based file was created to fulfill this component interaction, which is divided into four different sections. The definition of each section is located independent from others, meaning it could be placed in some point of the configuration file. The first explanation regards the number of steps that simulation have to perform to be considered done, the second takes into account the dimensions of the mapped area, the third is intended to explain the mapped regions in the environment, and the fourth refers to the vehicles that will be used in the simulation process.

Most of the simulation platforms executes in a period of time, not being associated a termination condition like resources depletion in a predator's prey environment, or certain value of profit in an economic process. Regarding the context of military warfare, the simulation should not terminate when all the agents of a team are captured or when an enemy invades a protected zone, but when the number of steps reaches the stipulated value. The XML example presented in 5.1 is intended to model the termination of the simulation process, using the <steps> tag within the <Simulation> tag.

As can be seen from 5.2, the dimensions definition is a simple tag that defines the how many cells will be used in the three dimensional modeled space. In the example, we are defining a region with 500 cells in the x-axis, 500 cells in the y-axis and 500 cells in the z-axis. All the following definitions of areas and vehicles should respect these boundaries for the good execution of the simulation process.

```
<Simulation>
  <steps>5000</steps>
</Simulation>
```

Listing 5.1: Simulation Definition

As presented in 5.3, the definition of areas has also a simple implementation that is very intuitive and easy to produce quick results. The example presented is intended to define a square land region with 40 cells in each edge. The tag for the areas definition is <Areas>, along with a single area definition tag <Area>. In each area tag there are a set of tags that are needed to be fulfilled. The first on is the type of the mapped area - <type> - which is land in the presented example. The possible set of constants for this definition is:

- **CELL_LAND**: definition of land region;
- **PROTECTED**: definition of protected region. The type of this area definition is related to validation processes, in which a certain region is not intended to be invaded by enemy team elements.

In the simulator we are not interested in defining the sea areas due to the default initial set of the whole area as sea, for further user's definitions of land areas. The simulator was conceived this way due to its maritime context, knowing that more sea region could be mapped against land.

The second set of tags that need to be filled is <point>. The tag point is intended to define a single vertex of the shaped area, and all the defined points will be connected in the forming the pretending area. The order of points is important for the linking shape process, not being independent from the place they are defined. In the example Listing 5.3 were defined four different points only using the x-axis and y-axis, making a plane that could be seen in the animation of the developed interface, since it only shows the two dimensional space of x-axis and y-axis.

The final tags of the XML-based file completion from the four needed refers to the vehicle definition. For the definition of several vehicles the tag <vehicles> is needed, along with the single vehicle definition tag <vehicle>. All the vehicles have some characteristics that should be gathered and implemented in the configuration file. Those tags are:

```
<Dimensions>
  <x>500</x>
  <y>500</y>
  <z>500</z>
</Dimensions>
```

Listing 5.2: Dimensions Definition

```
<Areas>
  <Area>
    <Type>CELL_LAND</Type>
    <Point>
      <x>10</x>
      <y>10</y>
      <z>0</z>
    </Point>
    <Point>
      <x>50</x>
      <y>10</y>
      <z>0</z>
    </Point>
    <Point>
      <x>50</x>
      <y>50</y>
      <z>0</z>
    </Point>
    <Point>
      <x>10</x>
      <y>50</y>
      <z>0</z>
    </Point>
  </Area>
  <Area>
    ...
  </Area>
</Areas>
```

Listing 5.3: Areas Definition

- **<team>**: this tag refers to the team definition for a single vehicle being the possible constants BLUE_TEAM or RED_TEAM;
- **<type>**: the type tag refers to the type of vehicle independent from the addressed team. The possible constants for this tags are: TYPE_PATROL or TYPE_NORMAL;
- **<goal>**: is the goal of the vehicle when it is initiated. This is an important attribute because it defines the behavior of the entity for the simulation, with the possibility of changing. The set of possibilities of this tag is flexible, since it is possible to add and remove constants from the structure file which possesses all the constants to be utilized in the simulation;
- **<behavior>**: this is the tag where the containing behavior information is addressed to a single vehicle. Using a specific model for behavior's implementation, the definition of strategies regarding a XML-based file is specified in 5.3.2. All the files that are associated to a single vehicle should be in a directory named "behaviors" in same folder of the application;
- **<speed>**: the measure of speed in the simulation and commonly used in the real world maritime velocity representation was the knot – 1knot = 1.852km/h. the maximum value for this parameter is 40, due to the most maximum speed of most patrol ships used in the military warfare and coherence in simulation process;
- **<radius>**: the tag radius is intended to specify the length of the circular perception that a single vehicle has from the environment. It could be applied either to an entity that is a vehicle (moving entity) or to a sensor (non-moving entity);
- **<coordinate>**: this tag specifies the location of the vehicle through the environment, which is supposed to be a valid coordinate. The initiation of the simulation will fail if the coordinate is malformed. For its definitions, only the x-axis and y-axis are considered into the present simulation platform.

The full version of the XML-based file could be seen in A with different modeled areas, different team vehicles, types and located in several places in the environment.

Validation File

For the simulator's platform, the validation file is not defined yet, since it depends on the DSS implementation that is an independent institution that is responsible for it. The main information of this file refers to the statistics of the simulation performed with the correspondent configuration file. The basic and mandatory information is the following:

- The number of defense vehicles;
- The number of strike vehicles;
- How many times a strike vehicle entered in a protected area without be seen;

```
<Vehicles>
  <Vehicle>
    <team>BLUE_TEAM</team>
    <type>TYPE_PATROL</type>
    <goal>GOAL_DEFENSE</goal>
    <behavior>beh_escort_patrol.xml</behavior>
    <speed>15<speed>
    <radius>40</radius>
    <Coordinate>
      <x>140</x>
      <y>110</y>
      <z>0</z>
    </Coordinate>
  </Vehicle>
  <Vehicle>
    ...
  </Vehicle>
</Vehicles>
```

Listing 5.4: Vehicle Definition

- How many strike vehicles were captured.

All the information is updated along the simulation's execution and generated when it terminates. Those parameters were provided by the SAFEPORT team and represent the pertinent information that should be considered by the DSS to calibrate its configuration file generation.

Structure File

For the organization and categorization of certain parameters like vehicle's goal or the performative's message, labels should be written and used for the interpretation of messages and behaviors between agents. One major example of its application is in the definition of behavior to each agent in the strategy XML-based file. Since the behavior's definition is dependent from parameters like perfomative's message, vehicle goal's or even which action should be performed, the usage of this type of file for the labels definition gains a total reason to exist. Hence, the structure file is a set of labels for the simulation platform to interpret, and the main tag is <struct>, and for each label definition should be used the <element> tag.

5.2.4 Class Model

The purpose of the class diagram is to show the developed classes for platform implementation, its attributes and methods, along with the interactions between classes, in a simple and easy representation. Since the programming language used to develop the application is Java, an object oriented and class-based, the class diagram, Figure 5.4 is the proper representation of the system architecture and each class will be explain in the following:

```
1
2  <Struct>
3    <Element>CELL_LAND</Element>
4    <Element>CELL_SEA</Element>
5    <Element>PROTECTED</Element>
6    <Element>BLUE_TEAM</Element>
7    <Element>RED_TEAM</Element>
8    <Element>UNDEFINED</Element>
9    <Element>TYPE_PATROL</Element>
10   <Element>TYPE_FRIGATE</Element>
11   <Element>TYPE_NORMAL</Element>
12   <Element>GOAL_DEFENSE</Element>
13   <Element>GOAL_STRIKE</Element>
14 </Struct>
```

Listing 5.5: Structure File

- **IO**: This class is responsible for the reading of XML files and its interpretation. Those files are the configuration file and strategy that could be associated to each vehicle. This class is also intended to create vehicles instances for the simulation usage, as well as the mapped region in which the agents should interact. The class uses the Document Object Model which is a language-independent that interacts with objects in HTML, XHTML and XML documents [Hor00], for the parsing of input files. The construction of vehicles instances lies in the further association with agents, along with its behaviors;

- **VEHICLE**: One of the main classes is the vehicle class, which is where all the information and characteristics of a vehicle is created. The characteristics that compose a single vehicle are: team; type of boat; initial, final and actual position; plan; goals; perception radius; velocity and associated behavior. This class is intended to be associated to each agent, being a real-world representation providing the agent the needed information for the correct environment interactions.

- **PLAN**: We've seen that plan is a constituent part of a vehicle. Hence, the plan is constituted by a set of tasks which could provide a complex behavior of the vehicle. The set of tasks is implemented by a LIFO (Last in First out) that is intended to be flexible in a way that if a certain new task appear and is added to the set, it would be immediately executed, and when it is terminated, the previous associated task could execute normally by the order they were inserted.

- **TASK**: A single task is just a certain path which has an initial and final point. The possible paths to be created are from two types: goto and survey. The goto path is just a straight line between initial and final points with no obstacles. If the calculated path has some obstacle that don't allow the direct connection between initial and final points, an alternative path is provided to circumvent it; the survey path is a more complex behavior constituted by circumferences and straight lines. A survey behavior is composed by a certain number

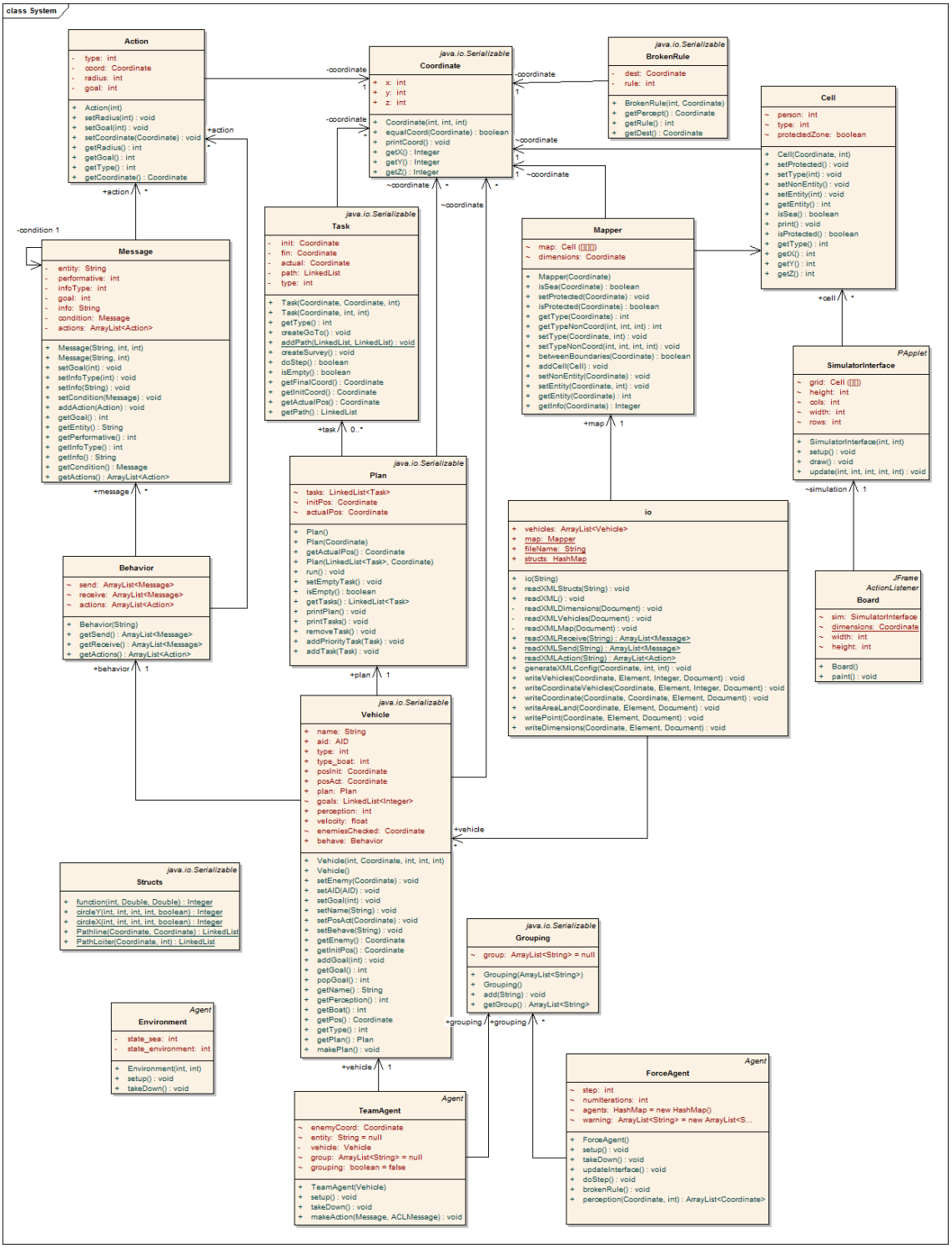

Figure 5.4: Class Model

of circumferences within a bigger circumference, which is created taking into account the provided radius for the bigger circumference. As can be seen from Figure 5.5, there are five

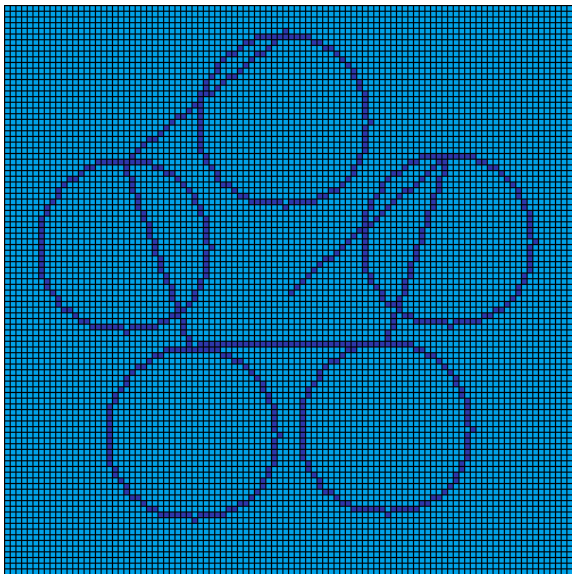

Figure 5.5: Survey Behavior

circumferences that are equally placed along the bigger circumference.

This is a typical behavior for the surveillance of a certain region varying with the previously defined radius.

- **BEHAVIOR**: The behavior class aims in create a flexible way of each agent know how it can act taking into account the interaction between agents. This class has two different types of message sets, one regarding the received messages, and the other the messages that should be sent. As agents could be reactive entities, each message as an action associated to. There's the possibility of an agent only receive a message without replying, for the purpose of inner state update – like agents localization – and has associated to it a certain action like goto or survey behavior. The same approach is applicable to the sent messages, which could induce a personal action. An agent could have initial actions that should be prioritized and begin without any interaction with the remaining agents. Hence, and regarding the XML files for the modeling strategy that each agent could associate, this approach is supposed to create a personal protocol that gives response to specific cases and creating an heterogeneous multi-agent system, where each agent has its own behavior.

- **MESSAGE**: This class is the definition of message that should be received and sent to different agents. A single message is constituted by an entity – agent that sent or should receive the message; performative – label that represents what type of message should be received; infoType – the type of information that is sent in the message; goal – the goal state that an agent should have to validate the message. This is important to validate due to the reception of same messages for different behaviors; info – the information associated to the message; conditions – this is a parameter that only exists in the sent messages. Most of the

times, a message is sent due to the reception of one another, and this condition parameter is from the same type of its own class - message – to ensure that the received message is validated for the sending of actual message; actions – this parameter is a set of actions that should be queue to an agent for the beginning of its own actions without the influence of other agents or even its own perception.

- **ACTION**: This is a simple class that only helps to organize the information of an action. An action is just a movement of an agent that could be one of the two previously explained tasks – goto and survey. This class is constituted by type - the type of the action (goto or suvery); coord – the associated coordinate that indicates the agent where to go. If the action is goto, the coordinate indicates the final point, being the initial point its actual location. If the action is survey, the coordinate indicates the central point of the bigger circumference survey task; radius – this parameter is only valid for the survey task, in which a radius is needed; goal – the goal that should be set for the inner state representation of agent. This class is used by the behavior class, in which represents the initial actions that an entity should perform, and by the message class, for the execution of certain behavior either if it's a receiving or a sending message.

- **COORDINATE**: This is a simple class that is intended to give a more flexible approach about the environment simulation. It is formed by three different parameters: x, y, z; that are the representation of a three dimensional coordinates. Despite its simple implementation, it is a very important class since almost the developed classes use this type of information for, e.g. location and path calculation.

- **CELL**: The class is part of the mapped region and constitutes the information of a single cell in the created board which is the environment representation. The cell information is composed by: person – which entity is in the cell; type – the type of cell (sea or land); protectedZone – indicator of protected zone cell. This information allows the map to be constituted by a set of cells, being easy its implementation and update according to the environment dynamics.

- **MAPPER**: For the environment representation, a class was created to promote the creation a single instance that is accessible to the entities. This representation is composed by a set of cells forming a three dimensional space, limited by the given dimensions of the mapped area. It's in this class where all the information about the environment is located, being changed with the environment dynamics, disregarding the dimensions of it.

- **BOARD**: For the user's interaction, the board class was implemented allowing the loading of configuration files, the simulation initiation, along with pause and stop. This is an important class not only due to its intermediate function between the application and user, but also for the simulator developer that needs some feedback from the modeling behavior work. The class is constituted by: sim – the animation panel representation; width and height which are the dimensions of the mapped region.

- **SIMULATIONINTER**: The animation of the simulation platform is implemented by the simulationInter class. This class uses an Processing Java interface that is implemented for the usage of the core functions which allow the good animation's execution. For this implementation, the following information is needed: grid: representation of the cell's animated board; height and width – the size of the representation panel; cols and rows – the number of lines and columns that the board should have.

- **BROKENRULE**: In certain situations, when the messages are exchanged between agents, there is the need of more information. The agents have rules that could be broken along the simulation execution, representing a change in the agent plans. The common occurrence of this broken is due to the perception of an enemy or an unknown object in the environment. This class is constituted by: dest – the coordinate that promoted the broken rule; rule – the violated rule.

- **STRUCTS**: This class was developed with the intent of aiding the other class implementation. The main functions are related to path calculation and obstacle avoidance. The importance of this class refers to the validation of the simulation, and as said before, the behaviors of agents and its proper replication in the simulator is a major indicator of good results.

- **GROUPING**: Is an auxiliary and simple class that guides the formation of a group. One important concept for the military context is the communication with a single group for the coordination of behaviors. This class as only a set of names, that are its participants and is used by the agents that need to know the partners of its own group for an adequate communication.

- **TEAM AGENT**: This class is an implementation of the Agent class of JADE, which provides a set of functions to override that guides the agent modeling. The team agent is one of the agents that constitutes the multi-agent system, and is intended to represent the vehicles that are used for the simulation. The parameters of this class are: enemyCoord – the coordinate of the perceived enemy. This variable is only filled if the vehicle if after a single enemy; entity – the main responsible of the group. All groups have an entity that controls and guides it; vehicle – the representation of the associated vehicle with all its characteristics; group – set of names that belong to a group; grouping – indicator of agent's participation in a group.

- **FORCE AGENT**: As all the agents, within a JADE agent-based platform, the force agent should implement the agent interface that JADE provides. This class is a representation of the agent that regulates the simulation of warfare, and communicates to all the environment integrated agents. The constitution of this class is: step – the time representation variable; numIterations – the maximum number of iterations for the simulation to terminate; agents – set of agent names for communication purposes; warning – set of agents names. The warning variable was conceived to regularly warn a set of agents for its own purposes, and it's activated by request from the team agents.

- **ENVIRONMENT**: This class is also an agent implementation of JADE middleware, and is intended to simulate the environment conditions like weather and sea dynamics. Hence, the variables used are: state sea – state of sea conditions; state environment – state of weather conditions. These states are regularly sent to the force agent for the influence update in communications and environment perception.

All the presented classes represent the system entities that allow the proper function and the functionalities that were modeled when the development of the platform. The interaction between classes is an important property of the system, in which classes use the representation of another's to build its own representation.

Environment

Regarding the environment that should be developed and replicated for a good simulation of real world military dynamics, the previously properties presented in Section 3 have to be considered. For a perfect simulation replication, is expected that vehicles have only local perception of the environment, and the actions to be performed over time, as a continuous space, not being executed like states in DFA. All the actions should contemplate the non-deterministic property, in which the executed actions could have different results, and the environment has the capability to change independently from the agent's action.

Some of the presented properties are difficult or almost impossible to replicate in computer simulation, due to the complex systems that composes the behaviors of objects to be modeled. The following description presents the implemented environment in the platform:

- **Accessible**: The modeled environment is accessible due to the local information that it provides to the agents. Agents can perceive from the mapped region, receiving information about what composes the area, knowing with some associated uncertainty the object located in some position;

- **Non-deterministic**: What leads to the description of a non-deterministic environment in the platform, is the associated uncertainty in the information given to the agents, begin revealed the proper information with a probability. All the other cases are deterministic due to the same action's repercussions in the environment as planned;

- **Static**: Despite the modeling of the environment agent be prepared for dynamic systems, the implemented property was static. This means that the environment has no capability to change independently from the agents' action. In other words, the environment has no direct influence in the agents' execution, not being changed if the agents have no capability to perform it. An extreme case, if agent have no actions associated, the environment would not change either, comparing with the dynamic parameter which allows the environment changing and even induce some action in the agents;

- **Continuous**: The implemented environment is continuous, being tasks and plan performed over time desregarding the state property defined as a DFA, characterizing the Discrete property. A plan could be reformulated any time, being reliable its execution over a continuous time, independent from others and with an associated duration.

As previously said, some properties are quite difficult to develop in computational simulation, being the presented properties the correct implementation for the platform requisites, regarding the simulation properties of the real world.

5.2.5 Multi-agent Architecture

Like in every agent-based platform a multi-agent architecture needs to be developed, defining the interactions between agents and its importance for the problem solving. The architecture should contemplate the projected relations between entities regarding the real world interpretation, ensuring the validation of the model and the consistency of the simulation process.

The multi-agent architecture used for the development of this application is based on the RSAC – Rand Strategy Assessment Center – System Software developed in 1985. It was developed aiming the improvement of strategy analysis methods, combining the best features about War Gaming and analytic modeling [Hal85]. Along with this system software, ABEL – Advanced Boolean Expression Language - was used for programming PLDs – Programmable Logic Device - developed in 1983 by Data I/O Corporation [Kyu85]. For RSAC application, it was used to write the decision rules of War Gaming context. The developed application is based in previously presented RSAC multi-agent architecture, with a slight change. Hence, the MAS of RSAC is composed by four different agents: Red Agent; Blue Agent; Scenario Agent; Force Agent. Comparatively to military life, the name of Red Agent refers to strike force, and the name of Blue Agent refers to defense force. The Scenario agent is responsible for simulating all the changes in the dynamic environment and the Force Agent combines all the previous agents into a simulation.

The changes of this architecture, that are presented in Figure 5.6, refer to the combination of Red and Blue Agents into the Team Agent. The purpose of merging these two agents stands in the similarity of agent architecture that both presents and ensure that the two teams are at the same level of modeling, not allowing the benefit of one of the teams, avoiding a biased simulation behavior, fulfilling the verification requirements for the simulation development process. In the nomenclature of RSAC architecture, NCL and ACL process from each Agent refers to National Command Level and Area Command Level. The first one is intended to create a plan based on the perception that is given to him and the conditions that lead to a plan change, and the second one refers to the implementation of the plan, sending it for Force Agent execution.

For the platform implementation, both NCL and ACL processes were developed, being the communication between the force agent and the team agent indirect sometimes and direct by others. In the multi-agent architecture was added a Data Dictionary intended to gather all the information from vehicles, mapped region and so on. This type of approach allows the reduction of exchanged messages between agents, speeding up the simulation time performance, gathering

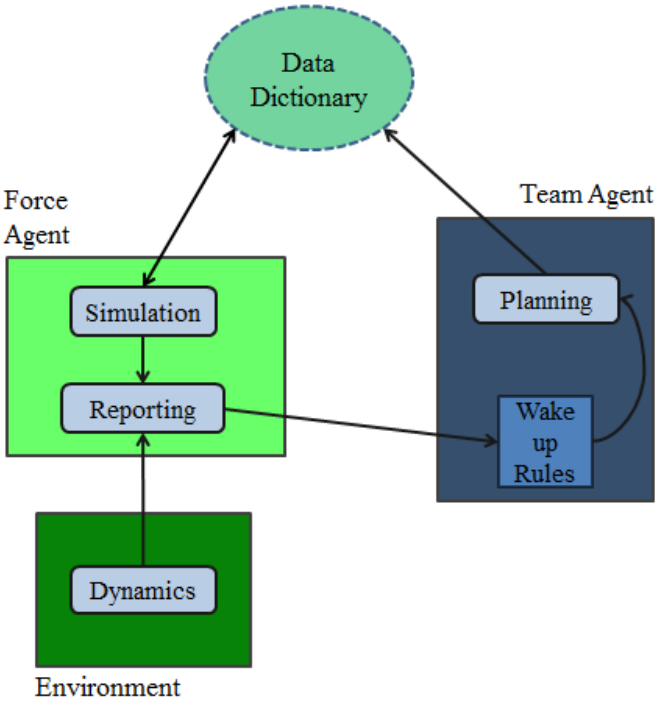

Figure 5.6: Multi-agent System Architecture

the information more quickly creating an indirect communication between agents. All the remaining information is gathered by messages exchange, using the FIPA-ACL messages that JADE middleware provides [Koe04].

The simulation is initiated when all the agents update its plans in the Data Dictionary so that force agent can gather the necessary information for the simulation to run. When the simulation is running, there are events that need to be reported for the team agents re-planning and suitability to the new environment conditions. The force agent communicates directly to the team agent reporting what was unexpected to happen that needs a reformulation of its behavior, which leads to the strategy update in Data Dictionary. This message exchange is called wake-up rule due the stand-by acting, from team agent, when the planning update and simulation execution. Regularly the environment agent sends the information about the environment dynamics to the changing of simulation conditions that influences the agent's interaction and perception. This influence is represented by the probability variation of enemy detection and maximum range perception reduction.

5.2.6 Agent Architecture

This section is intended to present each agent in detail and explain the main goals of each one. As said before, the multi-agent system is composed by three different agents that are capable of give response to the needs of simulation process.

- **Team Agent**: For this kind of simulation that implies the modeling of two different teams, an agent, which represents a single vehicle, is needed for its representation. In this case, by the entities similarity from different teams, the same type of agent is used implying that both strike and defense teams have the same capability to respond a certain modeling needs, developing strategies for each side not discarding the importance of validating the defense team. It's the team agent that is responsible for the strategy planning and given response to certain environment dynamics.

  Regarding the sensors definition, it's implementation is performed by team agent due to the flexibility of adaption to a modeling process. In this case, a sensor is just an object of a team agent that have the capability to percept, but it can't move along the environment, being only a platform that is always in loitering process. As well as the modeling of vehicles in the simulation platform, the sensor has the capability to communicate and report information that is important for the simulation process, and more specifically, for the team that is affiliated to.

- **Environment**: The modeling level of environment agent is quite simple, since its functions are the sea and weather updates to the force agents. This modeling was maintained simple due to the further adequate modeling of this agent from a specialized institution.

- **Force Agent**: The main function of this agent is to gather all the received information from environment and team agent, and apply it for the simulation execution. From the execution of a single step to the group forming process, this is the central coordination agent that guides the simulation of real-world and ensures that every received information is included and the final result is a consequence of it.

BDI software model

The BDI software model, agent's *beliefs desires* and *intentions*, is widely used for the implementation of intelligent agents, being a replication of real world behaviors regarding planning and guidance using goals to solve particular problems within the simulation environment [Bou04]. This type of software model is intended to guide the agent to a better problem solving, aiding in the selection of a proper plan, and the execution of it. The implemented agent that used this type of software model, is the Team Agent, which is the only that replicates the vehicles in the real world, being capable of planning and interaction.

- **Beliefs**: Beliefs are the conditions that constitutes the actual state about the world. The agent's beliefs cannot be mistaken as knowledge, due to the fact of sometimes beliefs are not the truth about, e.g. environment, but only *opinions* about what he sees. This property of the software model can be compared as the NCL of the RSAC multi-agent architecture that entails in the gathering of information, like the actual conditions properties, for the plan production;

- **Desires**: Desires can be seen as goals that are intended to simulate the motivation of an agent to execute a certain action. This motivation leads to what an agent should accomplish for the problem solving, being implemented in the platform like the inner representation of the agent's goal in its actual state. Those desires should guide the plan in its production phase, and it could influence which tasks should be performed, reaching the proper and adequate plan for its desires;

- **Intentions**: Intentions are the actual intention of an agent in performing an action when it has begun to be executed. In other words, is the motivation in the execution process and not before an action's performance, being capable of influence its continuity, leading to a plan reformulation or not. It can be seen as the ACL of the RSAC multi-agent architecture, in which all the information is sent for the plan's execution, implemented in the simulation platform.

With all the presented characteristics [Geo95], a proper and adequate software model for the agent's development was used, leading to an effective and efficient usage of agents in the first place, and the MAS for the second. This implementation is a good approximation of the real world execution, in which there are always goals that should be fulfilled and motivation over a plan execution, reaching a problem solving.

Communication

Like in the real-world communication, an understanding is needed for the correct interactions between entities. Living beings use multiple communication ways like speaking, writing, gestures, along with several ontologies which are represented by languages, being this way the usage of multi-agent system ontologies in the agent-based platform a need for the agents understanding and coordination. The communication between agents should provide the needed information for the correct simulation execution. To ensure it, there were implemented two different types of classes explained in Figure 5.2.4: brokenrule and grouping. Those two classes implement the Java serialization instead of using the ontologies provided by JADE. This is due to the simple and powerful means to convert Java objects into sequences of bytes, which is the type of information exchanged in ACL Messages [Bel07]. There are three main advantages of using ontologies provided by JADE instead of serializable classes:

- Independent from JADE agent implementation: Interpretation of messages between different agent implementation, since they are FIPA-compliant;

- Human-readable format when using the Sniffer agent for communication analysis;

- No need of knowing the object that an agent is receiving.

The advantage in the usage of ontologies instead of serialization by Java lies in the fact of no existence of other agents that could communicate to the simulation platform that are not JADE

implementations, but FIPA-compliant. All the agents that exist in the developed simulator are JADE implementations, so its interpretation of serializable classes is an easy and fast implementation, and there's no possibility to other external agents to join the simulator, making it a non-open system. For this reason, an agent knows previously what type of messages he will receive, being capable of understanding every message that he receives from the multi-agent environment. There is a possibility of communication with an external agent that represents the weather dynamics, but it is not implemented yet, and a possible ontology cannot be implemented until its definitions. Since the usage of Sniffer agent is much adequate in negotiation cases, for the protocol implementation of the simulator, the Sniffer agent was not used. The type of messages is not plural enough to justify its usage, and the implementation of the multi-agent system is intended to reduce the exchanged messages, giving emphasis of not using the Sniffer agent.

Hence, and for all the previous reasons, an ontology were not implemented, allowing a fast transactions of messages which is one of the more important requisites of all developed applications: temporal performance.

5.2.7 Interface

For the representation of simulation process, there are different approaches that should be considered when using a java implementation and a multi-agent system for simulation. Regarding the java implementation, there are GIS's (Geographical Information System) that provide a well and effective performance in the environment representation that could be integrated with Java implementations. A powerful toolkit developed by Open Source Geospatial Foundation is GeoTools [Jod11], which provides a full integration with java implementations, a real representation of the environment, and uses JTS_Topology_Suite Geometry. This type of toolkits is very useful since it is not necessary the implementation of an interface representation, but most of them are heavy for the system, being inappropriate for simulation means. One purpose that simulation implies is the fast feedback for real time usage, being a requisite for the development of this project. So, instead of using a GIS, a simple representation of the environment – essentially land and sea – was created, providing quick results for the animation process, not delaying the simulation. More important than that, is the non-mandatory animation representation of the simulation for the user, that devalues the interface that will be only used in specific cases, giving strength to the fact of not using a GIS with all the detail of the real world.

The simulator platform is composed by two different sections: animation and control panel. The first one refers to the animation of the simulation, which represents the movement of the vehicles allowing the user to see its execution in the environment, and the second one refers to the tools that allow the initiation, pause, stop and load of configuration files. The control panel as the basic user interaction needed for the well function of the simulator platform. Regarding the area fill problem, the most difficult issues found were algorithms that could handle large areas, and finding an inner point of the area mapped. Those were the main obstacles for a good representation of the mapped region provided by the configuration file. With the combination of those two methods, all the provided areas from the configuration file could be well represented and mapped in the user's

Figure 5.7: La Spezia, Italy mapped area

interface. For the area filling, the flood-fill algorithm was used, being an easy understanding and implementing algorithm, along with the ray-casting algorithm to find a polygon inner point. The final combination of those two methods can be seen in Figure 5.7, that uses different polygons with different shapes positioned in different sections of the environment.

The following subsections explain the methods used and present examples of its functions.

Flood Fill algorithm

For the areas that have the perimeter delimited with the final color of filling, the flood fill algorithm is a good and viable approach. The algorithm is based on a neighbors' structure, searching for cells that are not filled in the expected color, until non neighbor of the unfilled cells remains. The algorithm implementation uses recursion, meaning that the same function is called in itself. This method is used when the behavior of the function needs to be applied several times, depending on results of previous function calls. One of most well-known applications of the algorithm is for the factorial calculation. The recursive factorial function only has one stop condition: the function parameter, which is an integer higher than 0, being equal to 1; and for all the remaining cases, the parameter function is multiplied by the result of the same function call, with the parameter decremented in one. This leads to a number of the same function calls equal to the number given for the factorial calculation. The pseudo-code of the factorial calculation using recursion is presented in Listing 5.6.

```
Factorial(number):
  If number equal to 1, return 1
  Otherwise, return the number multiplied by Factorial(number-1)
```

Listing 5.6: Factorial calculation using Recursion

As can be seen from Figure 5.7, the function implementation includes the instruction that calls itself, using the same behavior for the well execution of the algorithm. Flood-fill algorithm starts with an inner position of the delimited area, the color to fill, and the color that should be replaced. With this information, the algorithm searches for neighbors that are no filled with the intended color, and calls the function again with the same parameters, using the updated inner position for the new neighbor cell.

The version of the algorithm that is presented uses a neighbor structure that only analyzes four neighbor cells (north, east, south and west), being possible the extension for eight cell searching (north, north & east corner, east, east & south corner, south, south & west corner, west and west & north corner). For the implementation of the simulator platform was used the four neighbor structure, due to cells' line that limits the area are linked sometimes by corner cell's, making the eight neighbor structure inadequate.

Most of the times, the mapped area should represent regions from the real world implying a very high number of cells that constitutes the simulated environment. Due to heap problems (dynamic memory allocation) using recursion, the space needed to keep the function calls in memory is higher than heap available space. So, a solution was found using queue data structure to store the functions' call information, which uses the stack, instead of heap, that has more available space. Regarding this changes, the pseudo-code of the final flood fill algorithm is presented in 5.8.

Ray Casting

```
Flood-fill (node, target-color, replacement-color):
If the color of node is not equal to target-color, return.
  Set the color of node to replacement-color.
  Perform Flood-fill (one step to the west of node, target-color, replacement-color
      ).
  Perform Flood-fill (one step to the east of node, target-color, replacement-color
      ).
  Perform Flood-fill (one step to the north of node, target-color, replacement-
      color).
  Perform Flood-fill (one step to the south of node, target-color, replacement-
      color).
Return.
```

Listing 5.7: Flood Fill four-way algorithm pseudo-code

```
1  Flood-fill (node, target-color, replacement-color):
2  Add the node the queue
3  While queue is not empty
4  Set the color of node to replacement-color.
5    If one step to the west of node has the target color
6      Add one step to the west of node to the queue
7    If one step to the east of node has the target color
8      Add one step to the east of node to the queue
9    If one step to the north of node has the target color
10     Add one step to the north of node to the queue
11   If one step to the south of node has the target color
12     Add one step to the south of node to the queue
13 Remove node from the queue's head
14 Return.
```

Listing 5.8: Flood Fill four-way algorithm pseudo-code

The ray casting algorithm was based on Jordan Curve's theorem, which says "any ray from inside a polygon crosses an odd number of edges on its way to infinity" [Hai90]. This means that if we start with a point that is out the filling surface, and if we trace a line through the polygon, an inner polygon point could be found between the odd and even crossed times, not the other way around. Figure 5.8 shows and example of ray casting method, in which could be seen that points between 1 and 2, and between 3 and 4 are inner points of the polygon.

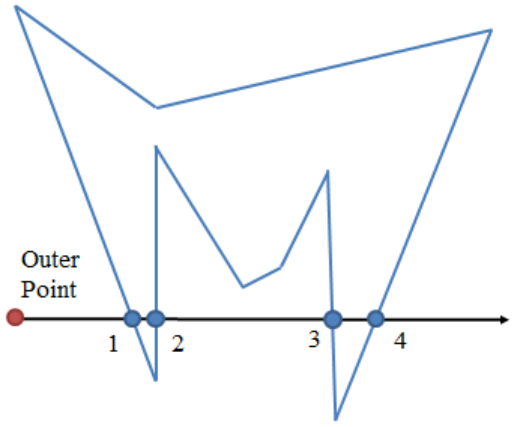

Figure 5.8: Ray Casting

The presented example shows that ray casting is an easy way of finding inner points from a polygon, but for the implementation of this method, a valid line that crosses the polygon should be calculated. Hence, it was used the maximum and minimum point of the polygon in the x-axis, being possible the calculation of the central region of the polygon. With this x-axis coordinate, which is for sure valid, all the y-axis coordinate are calculated creating a line that crosses the polygon. Figure 5.9 shows an example of the simulator implemented method.

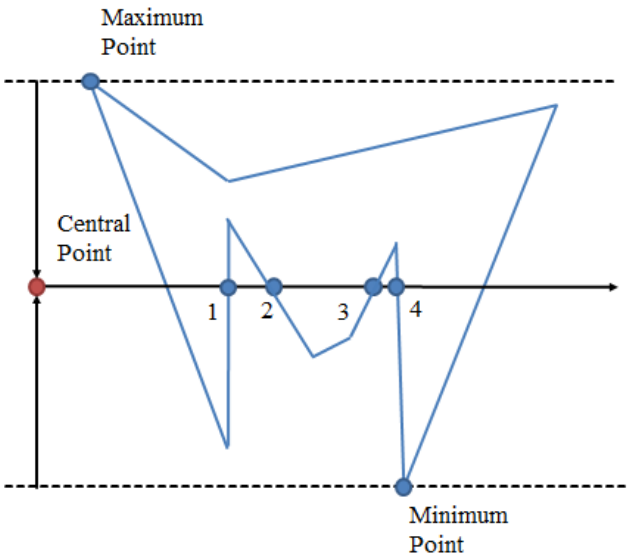

Figure 5.9: Ray Casting Implementation

SWING

As previously said, the developed interface is composed by SWING components which are intended to interact with the user, turning the simulation experience not only more appealing but also easier and effective. The created interface is composed by the normal interaction with a simulation environment – Start, Continue, Pause and Stop – allowing users to interact in a much richer way and control the simulation for analysis purposes. The user has the capability to load the configuration file to the simulator, enabling the execution of only the selected files, not predetermined ones, and run several different simulations in a certain order. For the user's guidance of the simulation's animation, information about the number of past steps is presented, useful for the analysis time comparison of actions and strategies. The final component that constitutes the user's interface is the slider simulation speed, which allows the user to manage the speed of simulation's animation. This component is useful for the selection of certain situations that need to be analyzed with slow motion detail. Figure 5.10 presents the created interface previously described, with La Spezia, Italy region model loaded.

The major limitations of SWING are related to performance issues, more specifically with animation and video representation. SWING uses its own handle to deal with the computer graphics, instead of using the native API's of the Operative System, e.g. DirectX on Windows. To correct this situation was used *Processing* for the simulation component's representation.

Processing

For the animation components of the simulator, the Processing programming language was

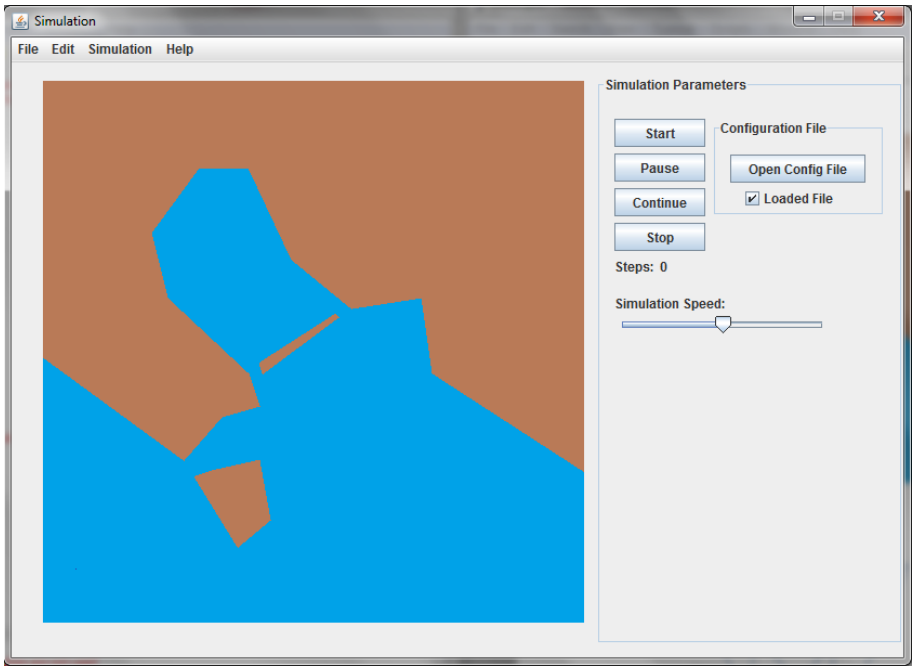

Figure 5.10: Simulator's Interface

used. It was mainly created for teaching computer programming through a visual context, to serve the electronic arts and visual design communities, being totally directed to representation and animation. It was totally written in Java, and all the Processing language is parsed to be translated into Java or Javascript, for Operative System applets or web-based programming, correspondently. The animation has a much better performance with Processing usage, instead of grid table, not only in the mapped region representation, but also for the vehicles presentation and animation. The animation is represented taking into account the strategies of each agent, and the mapped area considers the configuration file with the simulation properties.

5.3 Functionalities

5.3.1 Obstacle Avoidance

Regarding the piano mover's problem, several algorithms were developed to solve this type of problems. The problem is based on moving from an initial point to a final point, avoiding several platforms that are in the way of its trajectory. This problem can be extrapolated for a 2D approach, in which platforms are polygons, and trajectories are lines within a grid, simplifying the implementation of solutions to the problem.

Some well-known algorithms proposed for the piano mover's problem are Potential Fields [Kha95], Bug's algorithm [Ste90] or Brush Fire Algorithm [Cho04] which are used on the robotics for path planning calculation. The Potential Fields is an algorithm based on physics, in which the entity that is moving towards the environment is a particle that is attracted to its final

destination, and the obstacles represent a set of particles with repulsing properties. In every step of the moving process, the gradient is calculated to analyze in which direction the main entity should progress towards the final destination. This type of approach has a major problem due the inflexibility knowing if the particle stabilizes in a local position instead of the final one. Since the movement of the particle is dependent from the polygon's form, the concave shapes could limit the behavior of this algorithm, not being totally effective.

Another algorithm is the Brush Fire, based on neighbor structure to calculate the shortest path. The algorithm uses distances from the final path coordinate to the initial, determining which is the shortest path being totally independent from the obstacles that could exist in the environment. Since it uses a neighbor structure to progress, the algorithm needs to calculate all the neighbor cells until it reaches the initial coordinate of the path, being a problem when exists a high number of cells in mapped regions. The number of cells that the algorithm needs to analyze is very dependent from the location of the initial position, not having a constant performance. For the reasons previously presented, none of the algorithms was implemented due to temporal complexity problems. The information used for the path calculation in the simulator's platform differs from the previous approaches, since it provides the area's location and its constitution points, changing the paradigm of robot path planning which only uses the local information to calculate its route. Like the real world warfare, the information about the operation area is known using satellite communication or previous knowledge of the region, being totally proper and reasonable a personalized implementation despite of the local perception of the agents.

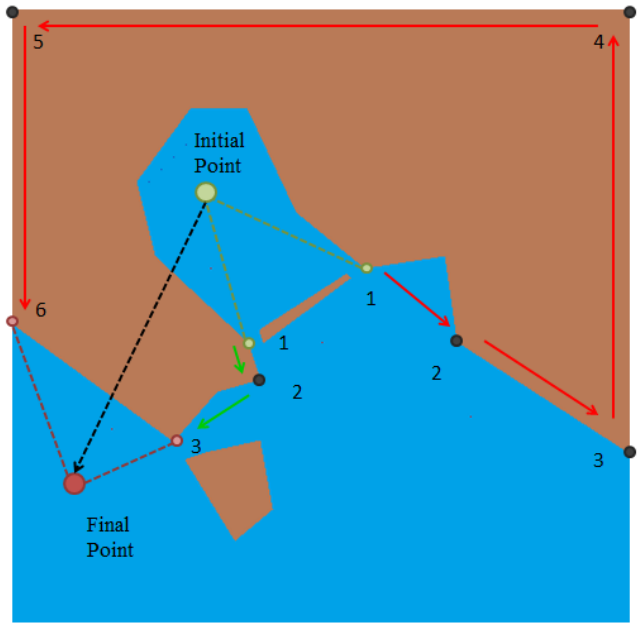

Figure 5.11: Example 1: La Spezia

The implemented algorithm takes into account the number of obstacle vertexes that a vehicle

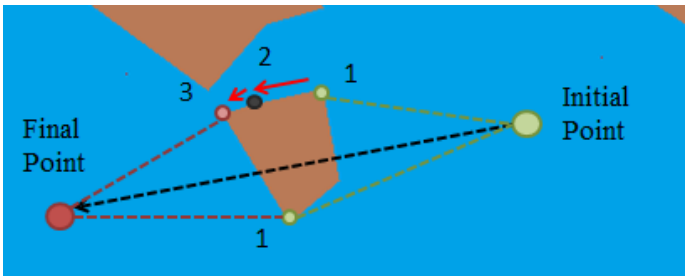

Figure 5.12: Example 2: La Spezia

should pass to reach the final coordinate. The algorithm is recursive since it needs to use the same behavior until the stop condition is verified. The first step of the algorithm is to know in which direction of the obstructed area it should move. To do that, the longest visible point of both initial and final are calculated, and the number of points between them are also calculated in clockwise and counterclockwise directions. To ensure the shortest path in both directions, the distance between the initial and final points of a single direction are calculated and compared, in which the next point in a certain direction is the longest visible point – no obstacles between - regarding the actual point.

The modeled area of La Spezia, Italy is taken as an example, and can be seen that the correct direction should be the lowest distance between the initial and final points, as it is presented in Figures 5.11 and 5.12

This algorithm ensures that correct orientation for the obstacle avoidance, along with the shortest path through the polygon. This approach combines the pertinent area vertexes that should be included in path planning, and the minimum distance through each direction calculation. The final product is the shortest path, simulating the normal action of the maritime forces in the real world.

5.3.2 Strategies

One of the main aspects of the platform's development regards parameterization, which allows changing the inputs according to users' will, reaching its goals. To give this type of flexibility taking into account the agents' behavior, a XML-based file was created. All the possible allowed simulator's behaviors could be modeled in a simple and effective way using the XML modeled files, which can be fully seen in A.2. The file's structure is segmented into three different sections: initial actions, receive message and send message. Those sections will be specified and explained with real world's examples giving a utility's overview of the platform independent modeling of behaviors. To fully understand and take better advantage of the modeling behavior, two different concepts should be explained: grouping and warnings. Those concepts are not mandatory, but represent a major aid for the behaviors implementation, allowing users to produce strategies in a more quality and easily way.

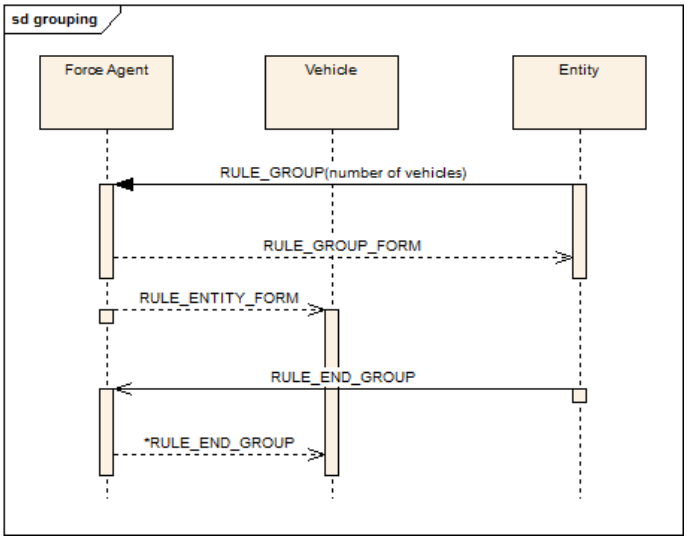

Figure 5.13: Grouping communication

Grouping

Most of operation and control systems need to coordinate and organize its entities for mission completion using the group properties to ensure a higher performance and effectiveness. To create the notion of group, there's always a responsible entity for the group's coordination, and the remaining entities that constitutes the group, that mostly receive orders and execute actions from the central entity. The workflow of a group constitution and messages that should be sent for the group's forming is presented in 5.13.

As can be seen, for the group's forming, an agent should require the service by sending a message, indicating the number of agents the group should have, to the force agent. The entities are available vehicles in which the goal is GOAL_DEFENSE, representing they are not, e.g. pursuing an enemy or in a group. The available agents receive a message with the entity that has the central role in the group, for the possibility of identifying who have sent the message. The main entity of the group receives all the names of the group's formation to communicate and give orders for what should be done and coordinate all the actions. To terminate the group, the main entity should inform the force agent sending a message with RULE_END_GROUP performative. This will induce a messaging from force agent to the group entities informing that they are no longer associated to the group. With this approach of force agent's regulation, the group formation can be controlled avoiding one entity belonging to several groups at the same time, as the main entity of the group. Hence, it is very important for vehicles that are intended to be available for this type of concepts to initiate with goal GOAL_DEFENSE and type of vehicle TYPE_PATROL in the configuration file. Otherwise, the force agent could not identify agents for the group's formation.

Also related to the grouping concept, there is a case that was modeled for the well function of the simulation, which regards the detection of the enemies in its perception radius. In case of

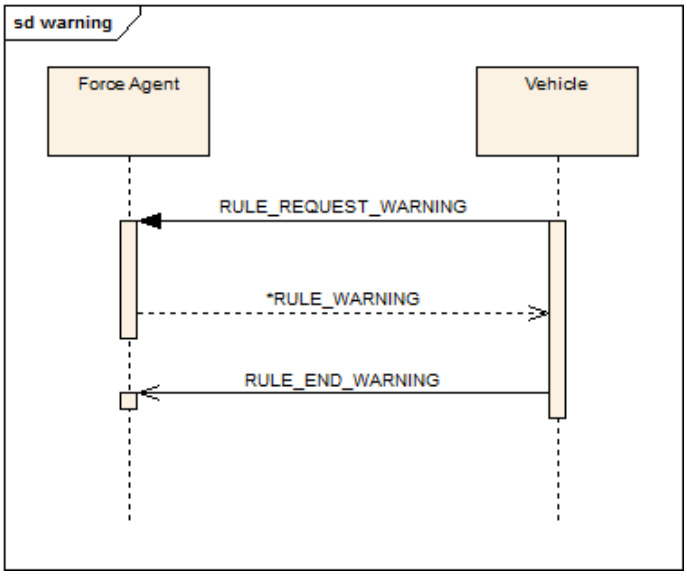

Figure 5.14: Warning Communication

enemy detection, the agent should contact the main entity of the group requesting authorization for enemy capture. Those behaviors are modeled in the software, and the acceptance of the main entity is associated with a fifty percent change. This value only represents the possibility for further usage of the simulator, being possible the modeling of associated probability.

Warning

Most of the times, entities need to promote a periodicity for informing purposes and do some state update of the vehicles. A good example is the usage of warning's process in group formation, in which an escort behavior should be done. The periodicity of informing the actual position of the escort entity to the patrol ships is essential, promoting an inner group state up to date with the exact coordinates of the escort vehicle, for the patrol ship to follow and provide the right defense means. The Figure 5.14 represents the work flow process of the warning service between agents.

The warning service should be requested to the force agent that regulates all the simulation, which implements the time component of the environment. The request message should only contain the RULE_REQUEST_WARNING performative for the service activation, and from the periodicity of 10 step simulation the agent should be warned to promote the modeled action that is associated to. Hence, to terminate the warning service, a message should be sent to the force agent, with the RULE_END_WARNING performative.

Initial Action

When the simulation is initiated, agents could have associated some actions that should be

```
<init_action>
  <type>SURVEY</type>
  <radius>25</radius>
  <coordinate>
    <x>100</x>
    <y>250</y>
    <z>0</z>
  </coordinate>
</init_action>
```

Listing 5.9: Agent's initial action example

executed without external influence, promoting pro-activeness in simulation process, when interactions are do not represent an important issue. The given XML example 5.9 represents a simple initial action of survey, explained in 5.2.4, in which survey action's radius, measured in number of cells, and the central coordinate should be specified. The possible constants for the <type> tag are: GOTO; SURVEY; GOAL. In case of GOTO constant usage, only the coordinate should be specified, like the given example, and for the GOAL constant, a <goal> tag should be added specifying the goal update for the agents. Any value of goal constant can be written, since it exists in the structure XML file.

Send Message

For the agents to send a single message to another's, creating this way a communication protocol and a way of interaction, some conditions should be verified, knowing if it is the proper message to send. An example of the condition XML code can be seen in 5.10, being constituted by <performative> – label that describes the type of communicative act; <sender> – the agent that sent the message; and optionally <goal> – goal associated to the agent that receives the condition message. The <goal> parameter is optional due to the necessity of sending messages independently from the agent's goal in cases that goal is not a constraint. The major strength of goal's parameter is in same message reception, with different agent's action, being a constraint for the correct reply choice. Within a condition, an action could be performed being an important parameter from the XML model since most of the message's sending is associated an execution constant: GOTO, SURVEY or GOAL. In case of GOTO in a condition parameters, multiple actions could be specified, allowing the user, e.g. to create a plan that is constituted by several tasks. The XML example in 5.10 is a representation of a simple condition to be associated with a *send message* modeled behavior.

Regarding the information of the message to be sent, it is composed by <to> – the receiver agent's name; <performative> – label that describes the type of communicative act; <infoType> – type of information to be sent. An example can be seen in Listing 5.11.

The constants to use in <to> label are restricted to *forceAgent* – the agent that regulates and

```
<condition>
<performative>RULE_ARRIVE</performative>
<sender>grouping</sender>
  <goal>GOAL_DEFENSE</goal>
  <action>
    <type>GOAL</type>
    <goal>GOAL_PATH</goal>
  </action>
  <action>
    <type>GOTO</type>
    <coordinate>
      <x>450</x>
      <y>20</y>
      <z>0</z>
    </coordinate>
</action>
</condition>
```

Listing 5.10: Sending message condition

performs the simulation; *entity* – the principal entity from the grouping concept; *grouping* – the whole group of the grouping concept, despite of central entity. Regarding group issues, the central entity can send message to the group using the *grouping* label, but the remaining entities of the group only can communicate to the entity using the *entity* label. The direct communication of agents is avoided due to the complexity of the XML files to promote a single and efficient communication protocol. If agents want to communicate, they should form a group and communicate between them. The <performative> tag accepts all the constants that the user inserts, since it exists previously in the *structure* XML file that contains all the labels to be used in the simulation process. Information could be added to a message, since its type is specified in the <infoType> tag. The accepted constants of this tag are: NON – no information associated; NUMBER – the information is a number, and is used for the group formation representing the number of needed agents; ACTUAL_COORDINATE – agent's actual position. If NUMBER constant is used, another tag must be added - <number> - for the number of agent's attachment to the message.

```
<send>
  <to>forceAgent</to>
  <performative>RULE_END_WARNING</performative>
  <infoType>NON</infoType>
  <condition>
  ...
  </conditon>
</send>
```

Listing 5.11: Send message

```
<receive>
  <from>forceAgent</from>
  <performative>CFP</performative>
  <infoType>NON</infoType>
  <action>
    <type>GOAL</type>
    <goal>GOAL_DEFENSE</goal>
  </action>
</receive>
```

Listing 5.12: Receive message

Receive Message

The structure of receive message's parameter on the XML file is similar to the condition's parameter. All used tags are the same, being also possible the user's definition of several actions to be performed, and an example could be seen in 5.12.

With all these three sections, the modeling of protocol communication and behaviors could be made, resulting in a strategy that could be associated to each agent independently. The major advantage of this approach is the adaptation that users can have to the real world strategy operations changing the agent's behaviors through simulation to simulation, being only necessary to change of XML associated files.

5.3.3 Simulation

Simulation process is a replication of the real world that is intended to analyze the produced dynamics to advice and promote the problem solving. Regarding the context of military warfare, some concepts should to be implemented for the proper execution of the simulation: Planning – each agent has an associated plan to execute; Perception – how interpretation of the agents should be from the world, and how it is replicated; Effectors - how defense vehicle should act in the presence of enemies; Process – the whole process of the simulation execution. All those concepts are explained in the underlying sections, being specified and dissected for a better user understanding.

Planning

Planning is the common process of an entity's organization consisting of several steps that should be performed to reach to purpose of its creation. It is commonly used in economy, development, project, etc., and also military campaigns, promoting a more effective and efficient approach to the problem solving process. The military behaviors are consisted in several tasks that could be performed leading to a global or transitory goal fulfillment. For the simulation implementation, planning is a set of tasks in a certain order and independent from each other, in which a

single task is a displacement through the environment. In this platform, the tasks independence is related to the none-necessity of next task's initial position being equal to the previous task's final position. This independence originates a new task that connects the existing gap between tasks. Has said in the early sections, the modeled tasks were the *goto* and *survey* that could be weave in a way that complex behavior can be modeled. When all the actions terminates, the initial actions modeled in the XML file associated to an agent are loaded creating a new plan for the vehicle. When an agent has a plan associated - set of tasks - and it percepts an enemy, a new set of tasks are added to the plan, not discarding all the previous ones. This approach was modeled with the intent of represent a transitory detour from the purpose he was intended, being self-regulated and capable of retour according to its basic modeled behavior.

Perception

The perception that each vehicle have from the environment was simulated like the real world implementation, using a circumference for the region to be analyzed. The radius of the circle is a parameter of the configuration file associated to each vehicle, and it could not be changed along the simulation. To set up a vehicle with no perception, the radius only needs to be zero, and not coordinates will be calculated for the case. The feedback that each agent receives from the environment perception is deterministic allowing the vehicles to conclude with certainty if an echo of the sensor is a vehicle from either same or opposite team, not having an associated probability of uncertainty. The perception of the agent is influenced by the environment conditions like weather and sea. The influence refers to uncertainty and the number or perceived cells from the environment, in which a worst condition from the environment is related to less number of perceived cells and less probability of knowing exactly the object that the echo returned from the sensor.

Effectors

The usage of effectors in real life is very difficult to model due to the unexpected behavior of the both defense and strike force. Hence, the modeled appliance of effectors was simply implemented simulating the capture of the enemy vehicle or sensor. The capture is based on the removal of strike force, and correspondent termination of simulation's agent, when a defense agent is near it. The modeled distance is 5 cells, and should simply replicate the usage of effectors in a simulation process.

Process

Since simulation is a replication of the real world dynamics, the temporal component has to be modeled in a way that its entities and the modeled environment evolves dependent from it. The common implementation of time simulation is the time step that represents the minimum range in which an action could be performed. It is presented has the minimum range due to the system

incapability to promote an action with the execution time lowest than the time step. Hence, all the actions are dependent from the time step, not being allowed to be performed faster than the simulation's replication. The step represents one second in real world, since the maximum speed of vehicles is 40 knot – 74 km/h – 20m/s, and a single cell represents 20 meters. So, in a case of 1000 steps of simulation duration, the real world proportion is more and less 16 minutes and 30 seconds. If the simulation is intended to simulate a whole week (7 days) of warfare, the number of steps is 604800.

In each step of the simulation, the planning of each agent is executed to simulate the displacement in the environment. Since the planning is a set of tasks, and a single task is defined by a path between two or more points, a step in a planning process is the next position of the vehicle in the environment.

Associated with the time-step that determines the time component of the simulator, the speed of the vehicles is other important component of the simulation, since interaction through environment like pursuing is performed. Hence, the correspondent speed of 1 cell per step in real world is 40 nods – 72 km/h, being the maximum speed of travelling independent from values above 40. For path planning of vehicles with a lower speed than 40 knots, the slowest speed is simulated with the repetition of the next coordinate. For example, if the specified speed was 20, the vehicle will remain in the same cell twice than a vehicle with a speed of 40 knots.

All the simulation's sections explained previously are an adequate approximation of the real world dynamics, from obstacle avoidance to the time step simulation, and even the implementation of the planning concept for the correct execution through the environment.

Chapter 6

Validation - Case Study

Tabletop exercises are composed by a set of scenarios that are intended to be simulated to test the response capability of an organization to a given event. Those exercises should simulate the environment dynamics with some fidelity and accuracy, for further reliability of the real world implementation.

The strong reason for the usage of tabletop exercises lies not only in testing the response capability of assets, but also for implementation validation. Those two purposes are the important previously explained Verification, Validation and Accreditation – Figure 2.2 – that should test the configuration file properties that are provided from DSS – Validation process – and the well development of the implementation to reach the best results – Verification process. The Accreditation process is dependent from the modular communication with the DSS, in which it should validate if the result should be good or bad, regarding the provided configuration file.

In the following section will be presented two different tabletop exercises for both Verification and Validation processes. The first one is intended to show a simple and small example of specific platform's capabilities, and the second is proposed to present a more complex situation, showing other platform's capabilities that could not be explored in the first exercise.

6.1 Tabletop 1

The presented case study is composed by: three members of the defense team (blue vehicles), which will perform the grouping, warning and authorization for enemy approach; and two members of the strike team (red vehicles) which only represent sensors, with no movement associated. Figure 6.1 represents the initial state of the simulation process, with the vehicles placed taking into account the configuration file, with the associated behaviors.

The main goal of this presented case is to create an escort situation from the initial location of the escorted vehicle to the lower left corner of the map, in which two available vehicles are selected for the task's completion. The enemy vehicles (sensors) are located in positions which allow the perception of the escort vehicles, and the authorization process is initiated for the sensors capture.

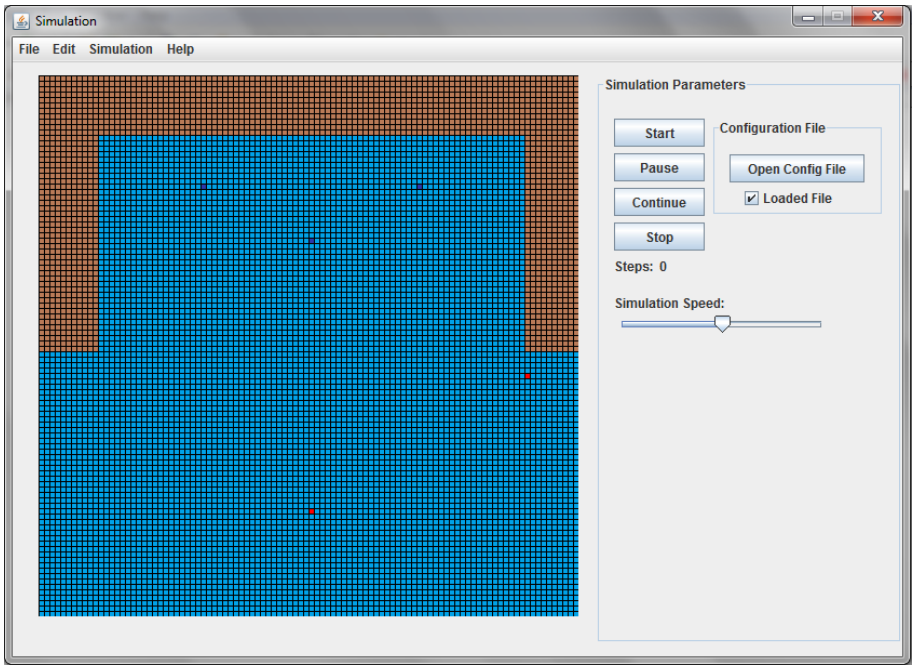

Figure 6.1: Case Study 1: Initial state

The underlying subsection is intended to explain how the described situation could be simply modeled and executed.

6.1.1 Configuration File

The modeled configuration file defines a dimension region as 100 x 100 cells, with simulation duration of 500 steps. The land region is constituted by clockwise orientation points that defines the final area representation – the brown filled cells - with a protected area located in the central sea parallelepiped region between land, presented in Figure 6.2.

Blue Team

Regarding the blue team members that are available for the escort process, the parameters are defined as Listing 6.1. The vehicle is defined as member of blue team - <team> BLUE_TEAM </team> - and its type is a normal patrol ship - <type> TYPE_PATROL </type>. For grouping purposes, which is the case, a vehicle only should be selected if its goal is set as GOAL_DEFENSE and type of vehicle as TYPE_PATROL. This allows the force agent to know which entities should select and inform for group formation. The perception radius and speed vehicle are defined as 20 cells - <radius> 20 </radius> - and 20 knots - <speed> 20 </speed>. The last parameter of the vehicle's definition is the location, in which a coordinate tag is used with the x, y and z-axis, being its initial position 30, 50 and 0. As previously said, the z-axis is discarded in this platform due to the scenario's representation using only two dimensions. For the remaining escort vehicles, only

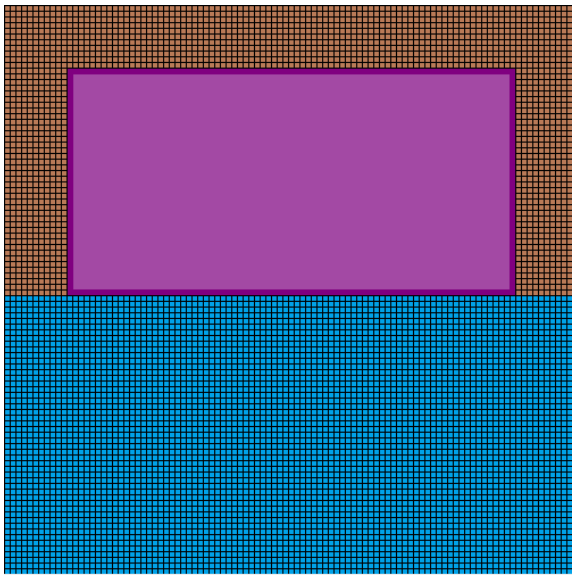

Figure 6.2: Case Study 1: Protected Area

the initial positions are different. The escorted vehicle has a behavior file associated, that regards the underlying description, which describes the way he acts through the environment, using the defined communication and interaction processes. The file definition is made using the behavior tag - <behavior> 1_beh_defense_new.xml </behavior> - and the specified strategy XML-based file should be placed in a folder called *behaviors*, and the folder placed in the root of the project.

```
<Vehicle>
  <team>BLUE_TEAM</team>
  <type>TYPE_PATROL</type>
  <goal>GOAL_DEFENSE</goal>
  <radius>20</radius>
  <speed>20</speed>
  <Coordinate>
    <x>30</x>
    <y>50</y>
    <z>0</z>
  </Coordinate>
</Vehicle>
```

Listing 6.1: Blue Team: Escort Vehicle

Red Team

As previously said, the members of red team represent sensors that only can perceive and has not any associated movement. For this matter, the specification of red team is the same as blue team's escort vehicles, despite of team member definition - <team> RED_TEAM </team> - and its goal

```
<Vehicle>
  <team>RED_TEAM</team>
  <type>TYPE_PATROL</type>
  <goal>GOAL_HOLD</goal>
  <radius>20</radius>
  <speed>0</speed>
  <Coordinate>
    <x>80</x>
    <y>50</y>
    <z>0</z>
  </Coordinate>
</Vehicle>
```

Listing 6.2: Red Team: Sensor Vehicle

- <goal> GOAL_HOLD </goal>. The definition of GOAL_HOLD regards the non-movement of the vehicle, being independent from its type definition. The modeling vehicle section is presented in Listing 6.2, and specifies the previously explained parameters.

The full version of configuration file is presented in A.

6.1.2 Strategy file

To be perform the grouping concept, a strategy file should be associated to the main entity of the group, which is responsible for its formation, coordination and termination. In this case study situation, the escorted vehicle the central entity of the group and might specify the number of vehicles to join, the location of the group vehicles and the authorization of group leaving. The presented strategy file is divided into two different sections: force agent communication; group communication. As explain previously in section 5.3.2. for the grouping concept to begin, a message should be sent to force agent - <to> forceAgent </to> - specifying the type of message - <performative> RULE_GROUP </performative> - the type of information sent - <infoType> NUMBER </infoType> - and the number of agents to join the group - <info> 2 </info>. Since this message is sent in the beginning of the simulation, the condition should be as follows - <performative> INIT </performative> - and sender - <sender> INIT </sender>. The full specification of message sending in presented in Listing 6.3.

The next step of the escort behavior that should be defined is the selected members' reunion near the entity. To promote this strategy a message should be sent for the group - <to> grouping </to> - to reunite - <performative> RULE_FOLLOW </performative> - near the central entity - <infoType> ACTUAL_COORDINATE </infoType>. For the group to go to a sent location, the RULE_FOLLOW should be used, not being possible the user's definition, like other's rules that an be specified in the structure file. The information type that should be sent is the actual location of the escorted entity, and it should be sent only when the force agent gather all the possible vehicles for the group formation. Hence, a message should be received from the force agent - <sender> forceAgent </sender> with the success of group formation - <performative>

```
<send>
  <to>forceAgent</to>
  <performative>RULE_GROUP</performative>
  <infoType>NUMBER</infoType>
  <info>2</info>
  <condition>
    <performative>INIT</performative>
    <sender>INIT</sender>
  </condition>
</send>
```

Listing 6.3: Group Formation Message

RULE_GROUP_FORM </performative>. The full specification of message sending to the group for reunion in presented in 6.4.

When the escorted vehicle initiates its way out of the protected area, it should inform the whole group of its location using the warning service that force agent provides. The message should be sent to the force agent - <to> forceAgent </to> - with warning service's message type - <performative> RULE_REQUEST_WARNING </performative> - that don't have any information associated - <infoType> NON </infoType>. To synchronize the message sent with the group arrival, a message from all the members - <sender> grouping </sender> - should be received with the proper message type- <performative> RULE_ARRIVE </performative>. In section 5.3.2, it was explained that an action can be associated with a message reception, allowing the behavior definition of the vehicle. In this case, when all the members of the group arrive near the escorted vehicle, it can initiate the displacement through the environment reaching the final location. Hence, the action is defined as a simple movement along the environment - <type> GOTO </type> - to its final position specified by x, y and z-axis to 85, 85 and 0, correspondently. The full specification of warning service initiation, group arrival and displacement for the final position is presented in Listing 6.5.

For the group update of the escorted entity, the warning service from force agent should be taken into account. To promote that, a message should be sent to the whole group - <to> grouping

```
<send>
  <to>grouping</to>
  <performative>RULE_FOLLOW</performative>
  <infoType>ACTUAL_COORDINATE</infoType>
  <condition>
    <performative>RULE_GROUP_FORM</performative>
    <sender>forceAgent</sender>
  </condition>
</send>
```

Listing 6.4: Group Reunion Message

```
<send>
  <to>forceAgent</to>
  <performative>RULE_REQUEST_WARNING</performative>
  <infoType>NON</infoType>
  <condition>
    <performative>RULE_ARRIVE</performative>
    <sender>grouping</sender>
    <action>
      <type>GOTO</type>
      <coordinate>
        <x>85</x>
        <y>85</y>
        <z>0</z>
      </coordinate>
    </action>
  </condition>
</send>
```

Listing 6.5: Warning Service; Group Arrival; Action Execution

</to> - with the proper type of message to follow - <performative> RULE_FOLLOW </performative> - and with the its actual position - <infoType> ACTUAL_COORDINATE </infoType>. The interpretation of the warning service is made by the reception of force agent's message - <sender> forceAgent </sender> - with the warning service label - <performative> RULE_WARNING </performative>. The full specification of group update is presented in Listing 6.6.

Regarding all the previously protocol specification, it should be terminated when the escorted entity arrives to its specified destination, and two messages might be sent: group deformation; warning service dissociation. Hence, when the entity's displacement terminates, the force agent informs it by sending a message - <sender> forceAgent </sender> - with the adequate message type - <performative> CFP </performative>. For the group deformation a message to the group should be sent - <to> grouping </to> - with proper deformation information type - <performative> RULE_END_GROUP </performative> - with no information associated. The main entity of the group is the only one that can dismember it, since it is the former. For the warning ser-

```
<send>
  <to>grouping</to>
  <performative>RULE_FOLLOW</performative>
  <infoType>ACTUAL_COORDINATE</infoType>
  <condition>
    <performative>RULE_WARNING</performative>
    <sender>forceAgent</sender>
  </condition>
</send>
```

Listing 6.6: Group Update

vice termination, a message to the force agent should be sent - <to> forceAgent </to> - with the warning dissociation' message type - <performative> RULE_END_WARNING </performative> - with no information associated - <infoType> NON </infoType>. The full specification of group dissociation and warning service termination is presented in 6.7.

```
<send>
  <to>grouping</to>
  <performative>RULE_END_GROUP</performative>
  <infoType>NON</infoType>
  <condition>
    <performative>CFP</performative>
    <sender>forceAgent</sender>
  </condition>
</send>
<send>
  <to>forceAgent</to>
  <performative>RULE_END_WARNING</performative>
  <infoType>NON</infoType>
  <condition>
    <performative>CFP</performative>
    <sender>forceAgent</sender>
  </condition>
</send>
```

Listing 6.7: Group Dissociation and Warning Service Termination

6.1.3 Simulation

The simulation is initiated like presented in Figure 6.1, and the strategy protocol starts to execute along with the simulation initiation. The first action that is performed is the reunion of the group around the escorted entity. Figure 6.3 represents the agents' position when they arrive near the request entity for grouping. All the requested entities for the group formation are displaced equitably along the main entity through a circle. In this specific case, since the number of request vehicles is 2, one of them is placed right above the escorted vehicle, and the other right bellow.

For the simulation process, the escort vehicles should follow the main entity of the group to its final location. The following process is presented in Figure 6.4 which can be seen the vehicles placed around the main entity. The actual positions of the escort vehicles regarding the escorted one are not the same, due to the delay of the force agent's warning service.

The group leaving for the enemy's sensor capture is a possibility if the main entity authorizes it. Figure 6.5 presents the situation when authorization is conceded to the two agents, and initiates the displacement to capture the sensors.

When all sensors are captured and the main entity arrives to its destination, the escort vehicles return to its initial positions, being the simulation of grouping, warning services and authorization process complete. For the appliance of this case study in other situations and modeled regions, the same method should be used, being guaranteed that the same result should be obtained. The

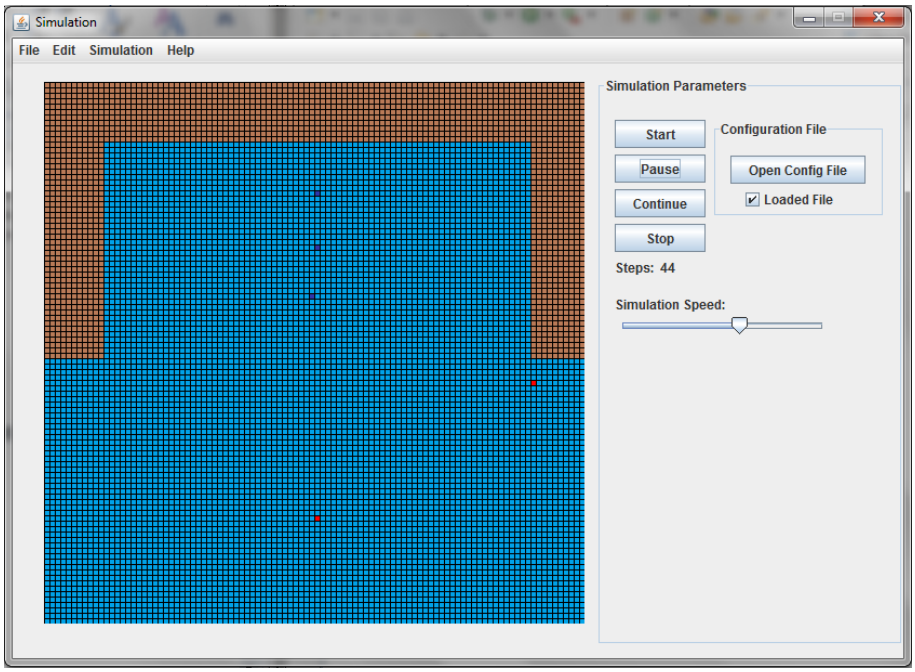

Figure 6.3: Case Study 1: Group Formation

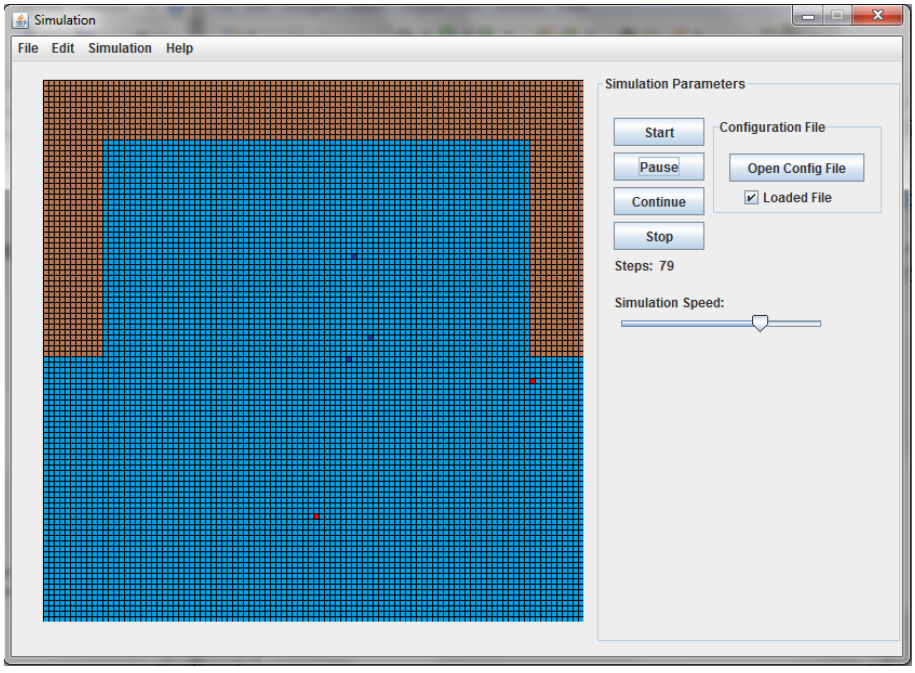

Figure 6.4: Case Study 1: Group Following

presented situations of this case study also can be executed when obstacles are presented, in which the shortest path is calculated and performed.

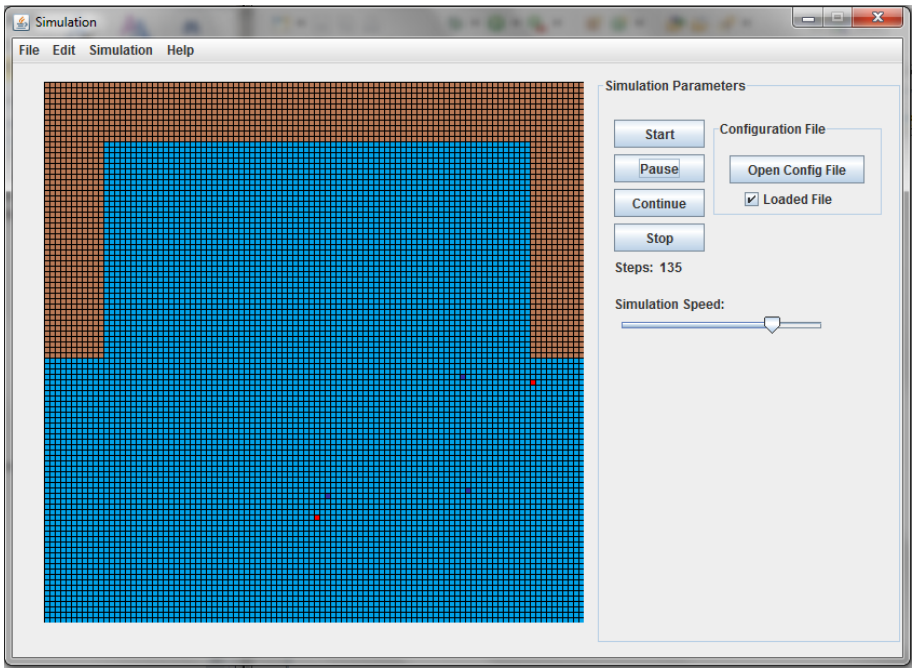

Figure 6.5: Case Study 1: Group Authorization

6.1.4 Results

The final results of the simulation process are presented in Table 6.1, in which the validation file should be created. As could be seen from the previous presented case study, there were three agents of blue team (defense agents), and two agents/sensors from red team (threats). Since red team entities were sensors and could not move through the environment, they didn't enter in the protected area, and like the authorization to leave the group was conceived for the group forming defense team, all the sensors were captured.

Table 6.1: Case Study 1: Results

Type	**Number**
Number of defense agents	3
Number of threats	2
Number of unseen threats	0
Number of captured threats	2

6.2 Tabletop 2

The following case study is intended to show a more adequate real-world situation, in which the defense force is placed in pertinent locations, and has to avoid the enter of enemies in the protected area. Is also intended to present the modeled properties of effectors appliance in the strike vehicles, and the final produced results for the validation file. The behaviors are not complex, focusing the

simulation in the basic strategies of entrance surveillance, not being explored the grouping or warning service presented in the previous sections.

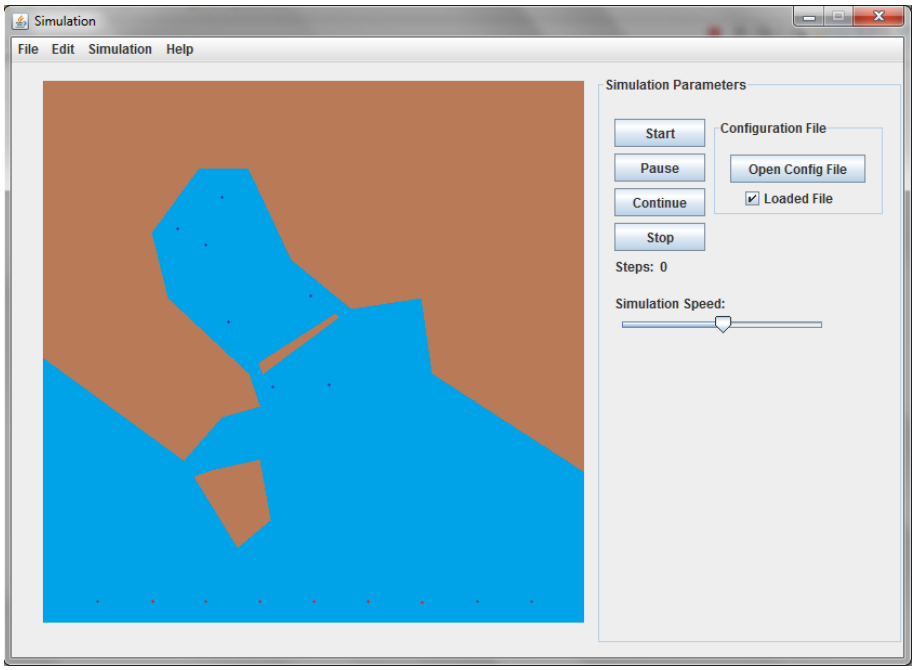

Figure 6.6: Case Study 2: Initial State

Other important validation of this case study is the obstacle avoidance of vehicles and the associated speed. The red team's vehicles are placed all over the y-axis the environment's bottom, to prove that a good implementation of obstacle avoidance was made, being reliable the simulation usage. For the simulation process a more complex configuration file was provided, with seven vehicles forming the defense force, and nine vehicles constituting the strike force or threats to the protected area. The initial layout of the simulation process is presented in Figure 6.6

6.2.1 Configuration File

As previously said, this simulation is intended to be more adequate for the real-world situation. Hence, the modeled region was La Spezia, Italy, being the dimensions of the environment 500 x 500, and the number of steps for the simulation to terminate is 2000. All the points that constitute the scenario were modeled by the clockwise orientation. A protected area, presented in purple, that was also modeled can be seen in Figure 6.7 and represents the real secure zone in the La Spezia's harbor.

Blue Team

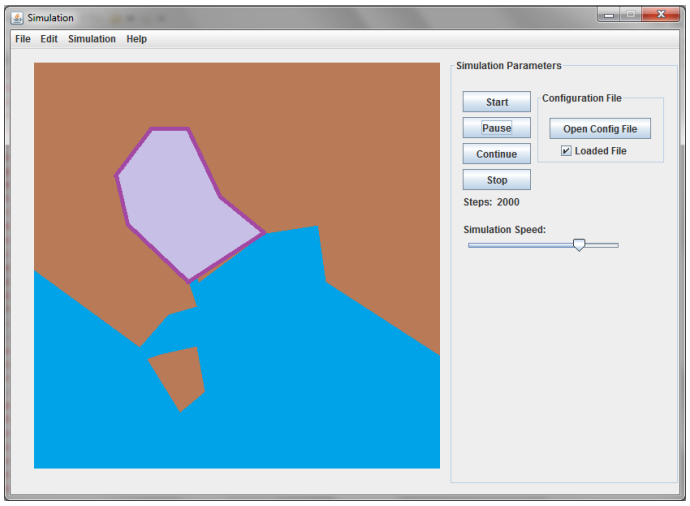

Figure 6.7: Case Study 2: Protected Area

The vehicles of blue team don't have a strategy file associated, being all the attack and enemy's capture modeled in the platform for the user's easier construction of the simulation model. The parameters of blue team are defined as 6.8.

```
<Vehicle>
  <team>BLUE_TEAM</team>
  <type>TYPE_PATROL</type>
  <goal>GOAL_DEFENSE</goal>
  <speed>40</speed>
  <radius>20</radius>
  <Coordinate>
    <x>282</x>
    <y>212</y>
    <z>0</z>
  </Coordinate>
</Vehicle>
```

Listing 6.8: Blue Team

Hence, the information of the configuration file regarding the blue team's vehicles is identical to Listing 6.1, despite of its location, speed and radius. Those three parameters have to be filled with the proper information of the modeled situation.

Red Team

Regarding the red team, a simple strategy file is associated modeling the displacement from its initial position to the core of the protected area and harbor. This strategy is intended to simply prove if the defense is prepared to deal with an invasion of the harbor, and certificates if the modeled implementation of effectors' usage is correctly made. The red team implementation is the

same as blue team, despising the team – RED_TEAM label should be used - and behavior tags. The behavior defines the strategy that a single vehicle should follow, which, in this case, all of them have a strategy file associated. The strategy file definition could be seen in Listing 6.9

```
<behaviors>
  <init_action>
    <type>GOTO</type>
    <coordinate>
      <x>150</x>
      <y>125</y>
      <z>0</z>
    </coordinate>
  </init_action>
</behaviors>
```

Listing 6.9: Simple Behavior

The provided coordinate refers to the core of the La Spezia harbor, and all the modeled red team's vehicles should move from its initial position to coordinate 150, 125 and 0 of x, y and z-axis, correspondently. To promote this approach, the initial type action of the vehicle should be GOTO, following by the correspondent final coordinate.

6.2.2 Simulation

When the simulation starts, all the vehicles are placed like Figure 6.6, and all of them have different types of speeds associated. So, the first feedback of the simulation process it the different type of displacement through the environment, and the different paths that were taken to reach the final location, as can be seen from Figure 6.8.

The next analysis that should be made is the capture of strike vehicles when a defense vehicle perceives from its defined radius, and entrance of strike agents in the protected zone. As can be seen from Figure 6.9, some of the strike vehicles were captured by the defense vehicles placed in the entrance of the harbor's protected zone, and a single strike vehicle entered in the protected zone. All these events will contribute for the final validation of simulation's output.

The final layout of the simulation is presented in Figure 6.10, in which can be seen that all the strike force was captured, and only one vehicle entered in the protected zone. From the left side menu can be also seen the 2000 number of simulated steps, which represents 33 minutes and 30 seconds.

6.2.3 Results

For the result analysis of case study's simulation, Table 6.2 was build based on the simulation results, and can be seen that all the strike vehicles were capture, and only one entered in the protected area.

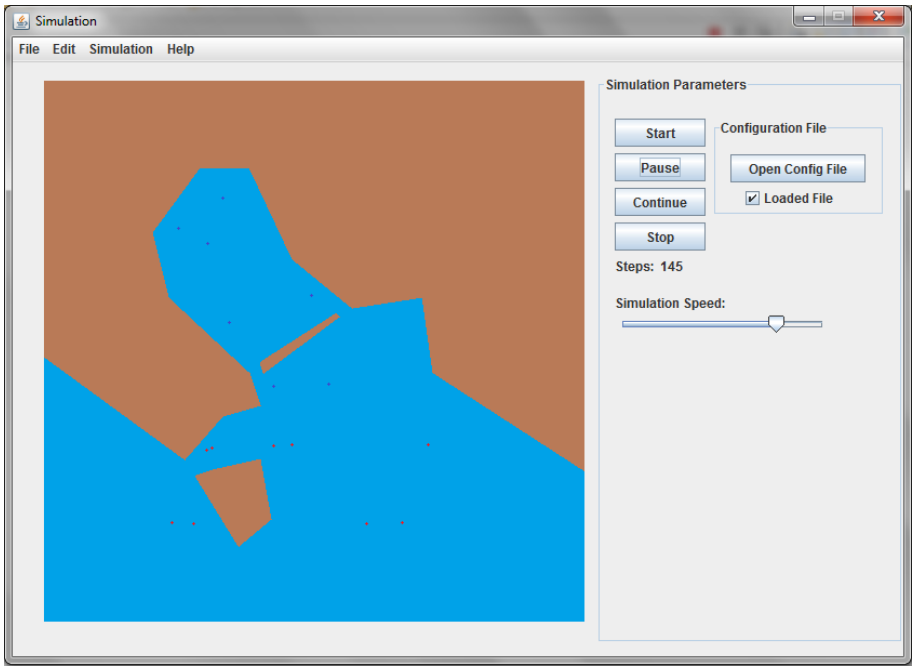

Figure 6.8: Case Study 2: Displacement of Strike Force

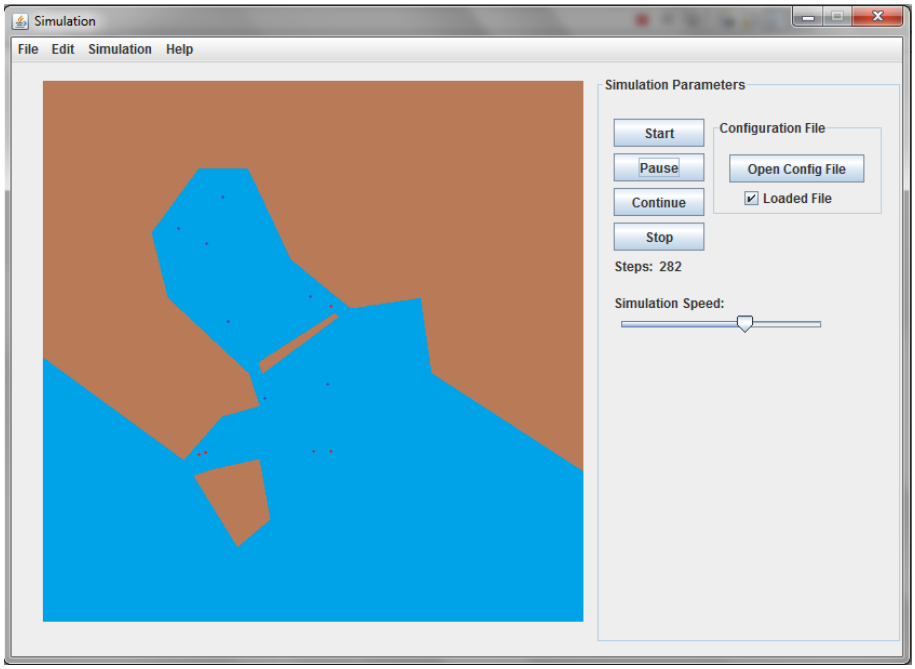

Figure 6.9: Case Study 2: Strike Force's capture and protected zone's entrance

Both presented cases are simple implementation that shows the potentials of the simulation platform as a warfare replicator, using two different teams' coordination and organization to promote the best performance in task completion and problem solving.

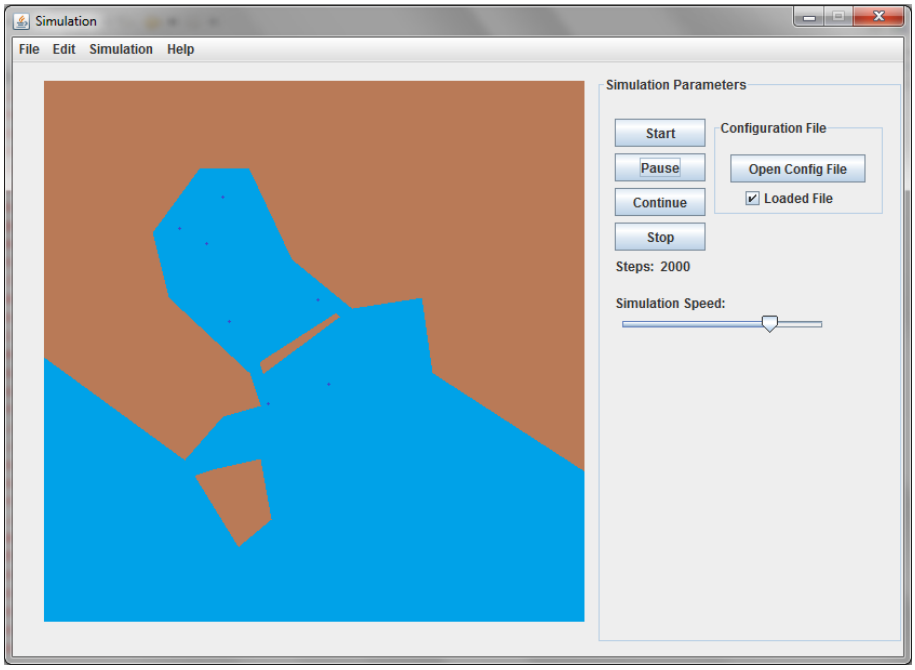

Figure 6.10: Case Study 2: Terminated Simulation

Table 6.2: Case Study 2: Results

Type	**Number**
Number of defense agents	7
Number of threats	9
Number of unseen threats	1
Number of captured threats	9

Chapter 7

Conclusion and Future Work

7.1 Work accomplished and conclusions

Regarding the planning for the dissertation's implementation, Appendix A, some aspects were fulfilled and some not. The scheduling of literature analysis and development of the project were followed and performed in time, aided by a *wiki* tool usage for the project details' registration and supervisor's meeting documentation. The writing of dissertation started later due to mandatory time for project's implementation, and the gathering of some reliable results that could validate the simulation's platform. Almost of the platform requirements were implemented in the simulator, providing a good and reliable approach for the real-world warfare simulation. The other functionalities that weren't performed are: Several simulation processes running at the same time; Simulation execution in different machines; Implementation of a Control Agent that could coordinate several simulations at the same time;

The execution of several simulations required a more deep study of the JADE framework, and a reformulation in multi-agent architecture for the inclusion of a new agent responsible for the coordination of simulation's executions, that provides to the DSS all the gathered information. It also implies a study of how strike forces should perform, in order to execute several simulations with different number of threats and behaviors that could validate, in a better way, the defense force. For the execution in distributed systems a deep understanding of the JADE implementation is also needed, being possible to develop a section responsible for it. One of the reasons that led to the JADE choice was the possibility to implement a system that can provide the proper feedback for functionalities that weren't developed in this project, taking also into account new implementation's possibilities.

As conclusion, the first aspect that should be underlined is the usage of agent-based system for simulation in general, and the warfare in specific. Most of the multi-agent system frameworks provide the proper tools for the correct and reliable implementation of simulation processes. Communication process, modeling of agent and multi-agent architecture, and distributed system's execution are a few number of the provided functionalities that can simulate an environment in which agents can organize, cooperate and compete.

The second aspect is the construction of complex dynamic systems using multi-agent systems. For the simulation process, the most common approach is the usage of differential equations that only can see the simulation process as a whole, and not specifically, reducing largely the exception treatment that real-world simulation requires. The sum of agents' behaviors and its interactions could perform a better implementation of the modeling, since is it based on simple linear modeling for each agent and the non-linear model emerge from the joining of these specificities. Even if the model don't behaves as expected, the calibration of the system is much more simple, due to changing of linear and simple behavior models, and not the approaches like differential equation that are complicated to solve.

Other aspect that should be noticed is the Processing programming language for multimedia approaches like animation, and the easy integration with other IDE's. Processing language is very simple and intuitive, which allows the fast implementation of interfaces that are very useful for the validation of modeled systems and simulation processes.

The final aspect is the good replication of the real world warfare that developed simulation platform provides. Functionalities like obstacle avoidance, group formation, interactive interface and either independent or flexible behavior definition were successfully implemented. In this dissertation, two case studies were presented and validated with different purposes and goals, and the whole software explanation from logical to physical architecture for the simulation process implementation.

Taking into account all the previous sections, can be concluded that the developed platform is a viable and reliable tool for the defense strategy analysis that can be fully integrated with the DSS, allowing the activation of the simulation processes in the NATO's SAFEPORT project.

7.2 Future Work

The future work composes an important part for the project to evolve, and many directions that could be explored for problem solving. The presented approaches regard the produced platform, and should be implemented taking into account the way of its implementation.

7.2.1 Artificial Intelligence

An important approach that could generate good results for the problem solving, is applying artificial intelligence methods for the combination of results. The genetic algorithms are known for the merging of at least two solutions, reaching a new one, based on the solution's codification. A single solution should be represented as a chromosome, in which the information is coded using an ordered set of genes, despite of its type of coding: binary, numeric, alphanumeric, etc. For the generation of new solutions, there are two different operators based on biology: mutation and combination (crossover). The first one regards the direct changing a solution by altering a single gene, and the second one is based on the combination of at least two solutions applying a several crossovers in different parts of the gathered solutions. Crossover is just a common point of separation of all the gathered solutions to produce a new one. Regarding the different parts of all

solutions resulting from the crossover's appliance, a single part of each solution is combined with the chosen parts of the remaining ones.

In this case, this type of approach could be applied as aid to the DSS generation of new configuration files. Several solutions proposed could be gathered and applied using a genetic algorithm to produce new and better solution performance, using combination and mutations operations. One aspect that might occur, is the generation of solutions that are not viable, representing a deficient set of genes that maps a not possible solution.

This approach could be one possibility in a huge number of artificial intelligence methods that could be applied for the solution reaching, and DSS aiding in the decision-making process.

7.2.2 Integration with DSS

Since the SAFEPORT project, as a whole, is composed by different independent modules, an integration process should align the global workflow, in which exchanged data have to be matched. A future work can be the improvement of both configuration and validation files for a better representation of information, e.g. by adding new parameters to XML-based files.

The definition of KPIs that can evaluate the solution's performance is also a goal to reach better results in the future. KPIs are intended to measure and evaluate the success or the particular activity success of an organization [FG90]. How a certain solution should be evaluated, is the responsibility of the developing company, with all the knowledge of the implemented system. This process is related to the Accreditation process – Figure 2.2 – in which the final behavior/result is enough precise for its usage in real life situations.

7.2.3 Sensors and Environment Improvement

The replication of sensors and environment is a great influence to the final result reliability. Sensors are replicated in the perception of each agent, which by default has a simple sensor that gathers information from the field. The implementation of different sensors that can be added or removed from each vehicle, is one real-world approach that should be used for the similarity warfare dynamics. The environment is also an important aspect that influences greatly the perception and communication of agents. The present implementation is based on none influence of environment in the simulation process, in which the perception of a single vehicle is always guaranteed that retrieves the whole information from the perceived area. The purpose is the environment representation by an external agent, that doesn't interact with the Team Agents, but with the Force agent which implements the simulation dynamics, being totally independent and could be treated as an external entity implemented, perhaps, in different programming language, since it is FIPA-complicant.

7.2.4 Strategies and Behaviors

The modeling of behaviors is in some way limited by the implementation of only two types of displacements through the environment. The production of new complex behaviors, that could

use the modeled displacement implementations, like new survey approaches or behaviors that take in account the whole group coordination, might help the user to model a simpler strategy XML-based file. The communication can also be improved using new communication parameters for the group coordination and organization in defense methods or even proper implementation of strike strategies, leading to a reliable and most real-world alike.

Appendix A

Appendix A

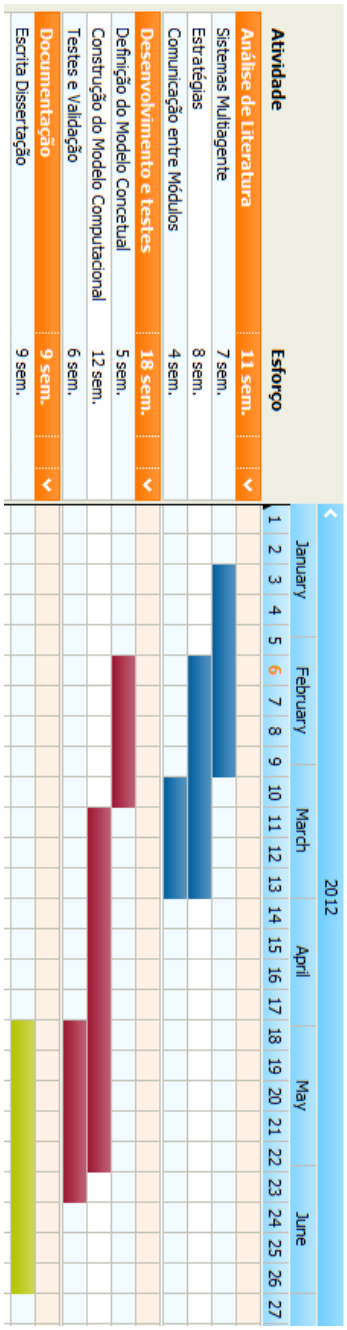

Figure A.1: Dissertation's Planning

```
<Map>
  <Dimensions>
    <x>500</x>
    <y>500</y>
    <z>1</z>
  </Dimensions>
  <Areas>
    <Area>
      <Type>CELL_LAND</Type>
      <Point>
        <x>0</x>
        <y>0</y>
        <z>0</z>
      </Point>
      <Point>
        <x>0</x>
        <y>499</y>
        <z>0</z>
      </Point>
      <Point>
        <x>360</x>
        <y>499</y>
        <z>0</z>
      </Point>
      <Point>
        <x>270</x>
        <y>360</y>
        <z>0</z>
      </Point>
    </Area>
  </Areas>
  <Vehicles>
      <Vehicle>
        <team>BLUE_TEAM</team>
        <type>TYPE_NORMAL</type>
        <goal>GOAL_OUT</goal>
        <behavior>behavior_file.xml</behavior>
        <radius>10</radius>
        <Coordinate>
          <x>150</x>
          <y>150</y>
          <z>0</z>
        </Coordinate>
      </Vehicle>
      <Vehicle>
        <team>RED_TEAM</team>
        <type>TYPE_PATROL</type>
        <goal>GOAL_HOLD</goal>
        <radius>15</radius>
        <Coordinate>
          <x>210</x>
          <y>160</y>
          <z>0</z>
        </Coordinate>
      </Vehicle>
    </Vehicles>
  </Map>
```

Listing A.1: Configuration File - First Part

```
<behaviors>
  <init_action>
    <type>GOTO</type>
    <coordinate>
      <x>100</x>
      <y>250</y>
      <z>0</z>
    </coordinate>
  </init_action>
  <send>
    <to>forceAgent</to>
    <performative>RULE_GROUP</performative>
    <infoType>NUMBER</infoType>
    <info>2</info>
    <condition>
      <performative>INIT</performative>
      <sender>INIT</sender>
    </condition>
  </send>
  <send>
    <to>grouping</to>
    <performative>RULE_FOLLOW</performative>
    <infoType>ACTUAL_COORDINATE</infoType>
    <condition>
      <performative>RULE_GROUP_FORM</performative>
      <sender>forceAgent</sender>
      <goal>GOAL_DEFENSE</goal>
    </condition>
  </send>
  <send>
    <to>forceAgent</to>
    <performative>RULE_REQUEST_WARNING</performative>
    <infoType>NON</infoType>
    <condition>
      <performative>RULE_ARRIVE</performative>
      <sender>grouping</sender>
      <goal>GOAL_DEFENSE</goal>
      <action>
        <type>GOAL</type>
        <goal>GOAL_PATH</goal>
      </action>
      <action>
        <type>GOTO</type>
        <coordinate>
          <x>450</x>
          <y>20</y>
          <z>0</z>
        </coordinate>
      </action>
    </condition>
  </send>
```

Listing A.2: Strategy XML-based file model - First Part

```
  <send>
    <to>grouping</to>
    <performative>RULE_FOLLOW</performative>
    <infoType>ACTUAL_COORDINATE</infoType>
    <condition>
      <performative>RULE_WARNING</performative>
      <sender>forceAgent</sender>
      <goal>GOAL_PATH</goal>
    </condition>
  </send>
  <send>
    <to>forceAgent</to>
    <performative>RULE_END_WARNING</performative>
    <infoType>NON</infoType>
    <condition>
      <performative>CFP</performative>
      <sender>forceAgent</sender>
      <goal>GOAL_PATH</goal>
    </condition>
  </send>
  <send>
    <to>grouping</to>
    <performative>RULE_END_GROUP</performative>
    <infoType>NON</infoType>
    <condition>
      <performative>CFP</performative>
      <sender>forceAgent</sender>
      <goal>GOAL_PATH</goal>
    </condition>
  </send>
  <receive>
    <from>forceAgent</from>
    <performative>CFP</performative>
    <infoType>NON</infoType>
    <action>
      <type>GOAL</type>
      <goal>GOAL_DEFENSE</goal>
    </action>
  </receive>
</behaviors>
```

Listing A.3: Strategy XML-based file model - Second Part

Appendix A

References

[AT90] Hendler J. Allen, J. and A. Tate. Readings in planning. *The Morgan Kaufmann series in representation and reasoning*, 1990.

[Bel07] Caire G. Greenwood D. Bellifemine, F. L. *Developing Multi-Agent Systems with JADE*. Wiley, first edition edition, 2007.

[Bou04] Le Page C. Bousquet, F. Multi-agent simulations and ecosystem management: a review. ecological modeling. *Ecological Modeling*, pages 176 – 313–332, 2004.

[Bra08] Paoli J. Sperberg-McQueen C. M. Maler E. Yergeau F. Bray, T. *Extensible Markup Language (XML) 1.0*. W3C Recommendation, fifth edition edition, 2008.

[Cho04] Lynch K. M.-Hutchinson S. Kantor G. Burgard W. Kavraki L. E.and Thrun S. Choset, H. Principles of robot motion. *MIT Press*, pages 86–88, 2004.

[CR07] Pereira A. Valente-P. Duarte P. Cruz, F. and L. P. Reis. Intelligent farmer agent for multi-agent ecological simulations optimization. *Springer-Verlag*, pages 593–604, 2007.

[DC89] Lesser V. R. Durfee, E. H. and D. D. Corkill. Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, pages 63–83, 1989.

[Fer95] J. Ferber. Les systèmes multi-agents. vers une intelligence collective. *InterEditions*, 1995.

[Fer96] N. Ferrand. Modelling and supporting multi-actor planning using multi-agent systems. 1996.

[FG90] C. T. Fitz-Gibbon. *Performance Indicators Pb*. Routledge, 1990.

[fIPA00] Foundation for Intelligent Physical Agents. Fipa acl message structure specification. 2000.

[Fra97] Graesser A. Franklin, S. Is it an agent, or just a program?: A taxonomy for autonomous agents. *Lecture Notes in Computer Science*, pages 1193: 21–36, 1997.

[Geo95] M. P. Georgeff. Bdi-agents: From theory to practice. *Proceedings of the First International Conference on Multiagent Systems*, 1995.

[Hai90] E. Haines. *Ray Tracing Ñews*. Volume 3, number 4 edition, 1990.

[Hal85] Shapiro N. Z. Shukiar H. J. Hall, H. E. *Overview of RSAC system software: a briefing*. RAND Corporation, first edition edition, 1985.

REFERENCES

[Hor00] Hégaret P. L. Wood L. Nicol G. Robie J. Champion M. Byrne S. Hors, A. L. *Document Object Model (DOM) Level 2 Core Specification.* W3C Recommendation, first edition edition, 2000.

[Ila04] A. Ilachinski. *Artificial War: Multiagent-Based Simulation of Combat.* World Scientific Press, first edition edition, 2004.

[Jod11] G. Jody. *GeoTools User Guide*. Manning, open source geospatial foundation edition, 2011.

[JPL97] Caltech Jet Propulsion Laboratory. Researchers stage largest military simulation ever. 1997.

[Kha95] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, pages 90–98, 1995.

[Koe04] Nourbakhsh I. Sycara K. Koes, M. Communication efficiency in multi-agent systems. *In Proceedings of ICRA 2004*, pages 2129–2134, 2004.

[Kyu85] L. et al. Kyu. A high-level design language for programmable logic devices. *Manhasset NY: CPM Publications*, pages 50–62, 1985.

[Lan56] F. W. Lanchester. Mathematics in warfare. *The World of Mathematics*, pages 2138–2157, 1956.

[Lot20] A. J. Lotka. Analytical note on certain rhythmic relations in organic systems. *Proceedings of the National Academy of Sciences of the United States of America.*, pages 6: 410–415, 1920.

[MAK11] VT MAK. Vr-forces: The complete simulation solution. 2011.

[Nor06] N.T.; Vos J.R. North, M.J.; Collier. Experiences creating three implementations of the repast agent modeling toolkit. *ACM Transactions on Modeling and Computer Simulation*, pages 1–25, 2006.

[oD98] Department of Defense. *Department of Defense Modeling and Simulation (M&S) Glossary*. DoD, 1998.

[Oli99] E. Oliveira. Applications of agent-based intelligent systems. *Proceedings of 4th SBAI-Brasilian Symposium of Intelligent Automation*, 1999.

[oPH05] Harvard School of Public Health. Largest computational biology simulation mimics life's most essential nanomachine. *Los Alamos National Laboratory*, 2005.

[Per10] Correia A. M. Pereira, A. *Intelligent simulation of coastal ecosystems*. PhD thesis, Faculdade de Engenharia da Universidade do Porto, 2010.

[Rea07] Fry B. Reas, C. *Processing: A Programming Handbook for Visual Designers and Artists.* MIT press, first edition edition, 2007.

[RV03] M. Robinson and P. Vorobiev. *Swing*. Manning, second edition edition, 2003.

[Sar91] R. G. Sargent. Simulation model verification and validation. *Proceedings in Winter Simulation Conference'91*, pages 37–47, 1991.

[Smi98] R. D. Smith. Essential techniques for military modeling and simulation. *Proceedings of Winter Simulation Conference' 98*, pages 805–812, 1998.

[Ste90] V. Stepanov, Lumelsky. Path-planning strategies for a point mobile automation amidst unknown obstacles of arbitrary shape. *Autonomous Robots Vehicles*, pages 1058–1068, 1990.

[Tay83] J. G. Taylor. Modeling and simulation of land combat. *Georgia Institute of Technology*, 1983.

[Vol26] V. Volterra. Variazioni e fluttuazioni del numero d'individui in specie animali conviventi. *Accademia dei Lincei*, pages 2: 31–113, 1926.

[Woo95] Jennings N. R. Wooldridge, M. J. *Intelligent agents: Theory and practice*. Knowledge Engineering Review, 1995.

[Woo02] M. J. Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, second edition edition, 2002.