FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Cooperative Multi-Robot Missions: Development of a Platform and a Specification Language

Daniel Augusto Gama de Castro Silva

Programa Doutoral em Engenharia Informática

Supervisor: Luís Paulo Gonçalves dos Reis (Prof. Dr.)

Co-Supervisor: Eugénio da Costa Oliveira (Prof. Dr.)

2011, June

To my parents and my grandmother

Abstract

In the past years, the number of autonomous vehicles developed for performing different tasks has increased significantly. Both in military and civilian scenarios, vehicles with an increasing level of autonomy are used to execute missions that take place in a location that is either hazardous or inaccessible for humans. They are also used when it would be impracticable to use vehicles operated by humans, either due to the strain on human resources caused by long-term missions or because it would simply be too costly to use such resources. Many projects have been developed that make use of such vehicles for a variety of purposes. Most of these projects use a limited variety of vehicles and/or are designed to be used for a narrow range of missions. Most of the specifics for each platform are also usually defined in an ad-hoc manner.

This dissertation aims at providing an integrated platform for the execution of a wide range of joint missions by a group of heterogeneous robotic vehicles. For that, both a platform and a set of languages that allow for the configuration of a mission have been developed.

In a first phase, a general architecture for systems that include autonomous vehicles was designed, taking into consideration several common requirements. This general model was then instantiated into a more specific architecture, where several components interact to allow for the definition and execution of cooperative missions.

After designing the platform architecture, and in order to configure all aspects regarding the mission, four XML-based dialects were created – scenario, teams, disturbances and missions. These dialects were categorized as static vs. dynamic and also according to their orientation toward the operating scenario or the team. The Scenario Description Language (SDL) describes the static elements of the scenario in which the mission will take place. The Team Description Language (TDL) describes the composition of the team and some team-specific constraints.The Disturbances Description Language (DDL) describes all (dynamic) elements anomalous to the environment that will constitute targets for the missions.The Mission Description Language (MDL) describes a mission for the team to perform, constituted by a set of phases.

Finally, a platform implementing the defined architecture was developed, comprised of several components, including a realistic simulator; a control panel for the operator to specify the mission and related elements; an agent to control traffic in a centralized manner on a localized area; an agent to control each vehicle, responsible, among other things, for mission planning and execution; and some other components.

This implementation of the platform and languages allows for a flexible configuration of missions, team and mission scenario. By using a test setting that included a scenario configured with two airports, a team comprised of several aircraft, two simple disturbances and a simple recognition mission, several components were tested when working together, in order to validate the approachTests with other vehicle types also proved the feasibility of using this platform with vehicles other than aircraft, showing that the approach is valid and the platform can be used with different vehicle types. Also, the platform's modularity and flexibility allows it to be used as a testbed in several new research directions.

Resumo

Nos últimos anos, o número de veículos autónomos desenvolvidos para realizar diferentes tarefas aumentou significativamente. Tanto em cenários militares como civis, veículos com um nível de autonomia crescente são usados para executar missões que se ralizam em locais que são ou perigosos ou inacessíveis para seres humanos. São também usados quando seria impracável usar veículos operados por seres humanos, devido à tensão nos recursos humanos causado por missões de longo-termos, ou simplesmsnete porque seria demasiado custoso usar tais recursos. Muitos projectos foram desenvolvidos para usar veículos autónomos para uma miríade de fins. A maioria destes projectos usa uma variedade de veículos limitada, e/ou estão desenhados para ser usados num conjunto limitado de missões. A naioria das especificidades de cada plataforma são também normelmente definidas de uma forma ad-hoc.

Esta dissertação tem como objectivo fornecer uma plataforma integrada para a simulação e execução de um vasto conjunto de missões por um grupo de veículos robóticos móveis heterogéneos. Para tal, foram desenvolvidas tanto uma plataforma como um conjunto de linguagens que permitem configurar a missão a ser executada.

Numa primeira fase, foi desenhada uma arquitectura genérica para sistemas que incluem veículos autónomos, tendo em consideração vários requisitos comuns a estes sistemas. Este modelo genérico foi depois instanciado numa arquitectura específica, em que vários componentes interagem para permitir a definição e execução de missões cooperativas.

Após o desenho da arquitectura da plataforma, e de forma a permitir a configuração de todos os aspectos relativos à missão a ser executada, foram criados quatro dialectos baseados em XML – cenário, equipas, distúrbios e missões. A Linguagem de Descrição de Cenário (SDL – *Scenario Description Language*) descreve as bases de operações existentes, de onde os veículos poderão operar (as bases de operações podem incluir um aeroporto, porto, e/ou base terrestre), tipos de veículos (que serão instanciados pela equipa de veículos), e estruturas globais de controlo – áreas de voo restrito e controladores de tráfego para as bases de operações. A Linguagem de Descrição de Equipa (TDL – *Team Description Language*) descreve os veículos que compõe a equipa, assim como algumas restrições específicas da equipa, tais como as bases de operações que a equipa pode usar ou áreas de voo restrito adicionais. A Linguagem de Descrição de Distúrbios (DDL – *Disturbances Description Language*) descreve todos os elementos anómalos ao ambiente, e que irão constituir alvos para as missões (incluindo seres vivos, fenómenos naturais tais como fogos, eventos artificiais ou veículos). Os distúrbios podem ser criados num ponto espacial e temporal específico ou aleatório, sendo estacionários ou móveis, com um número variável de componentes, cada um deles com tamanho fixo ou variável e requerendo diferentes sensores para ser detectado. A Linguagem de Descrição de Missões (MDL – *Mission Description Language*) descreve a missão como um conjunto de fases (que podem ser dependentes umas das outras), cada uma realizando-se em deteminadas áreas, e contendo possíveis requisitos e dicas, que funcionam como restrições rígidas e flexíveis para a fase, respectivamente. Além disso, cada fase tem um número de alvos, os quais podem variar em multiplicidade, e de acordo com o distúrbio que se relaciona com o alvo.

Finalmente, foi desenvolvida a plataforma que implementa a arquitectura definida, constituída por vários componentes. Depois de realizar uma análise de quatro categorias de alguns simuladores de voo existentes, o Microsoft Flight Simulator X foi escolhido pelas suas características, e provou ser um simulador realista, fornecendo tanto uma simulação física do ambiente e veículos como uma representação visual da simulação. Para a especificação de todos os elementos referentes à missão, o Painel de Controlo (*Control Panel*) fornece uma interface gráfica para o operador especificar todos os elementos definidos nos quatro dialectos criados e controlar a plataforma. Um Agente de Controlo de Tráfego Aéreo (ATC – *Air Traffic Control*) foi desenvolvido para controlar o tráfego de forma centralizada numa área localizada. Este agente estende o uso desta plataforma à investigação de controlo de tráfego. Um Agente de Controlo de Veículo (*Vehicle Control Agent*) representa cada veículo, e tem várias responsabilidades, incluindo planeamento e execução de missões. Um Gestor de Distúrbios (*Disturbances Manager*) é responsável por representar todos os distúrbios no ambiente quando o simulador não o pode fazer, interagindo com os veículos e fornecendo-lhes leituras simuladas dos sensores. Uma Ferramenta de Análise de Desempenho (*Performance Analysis tool*), baseada nos ficheiros produzidos pelos diversos componentes da plataforma, e usando um conjunto de métricas adequado a cada tipo de missão, pode fornecer uma análise e informação de desempenho de uma missão ou conjunto de missões.

Esta implementação de plataforma e linguagens permite uma configuração flexível de missões, equipas e cenários para as missões.

Résumé

Ici le résumé en Français.

Acknowledgments

I would like to start by expressing my gratitude towards my advisors, Luís Paulo Reis and Eugénio Oliveira, for all the support they gave me over the last years, for the discussions that helped me with my work, for all the ideas that came from those conversations, and also for their friendship. I would also like to thank João Tasso Sousa, Luís Antunes and António Augusto de Sousa for the valuable comments provided in the initial doctoral committee meeting.

My thanks also to Andrea Passadore for his help with the AgentService platform, and to all who helped develop this platform. A thank you note also to the team that developed Flight Simulator X, a great game that has provided me many hours of joy, and also a great simulation environment that has given me many hours of hard work. I would also like to thanks João Correia Lopes for making available the LaTeX2e template used in this thesis, developed by himself and João Canas Ferreira.

My thanks also to all who collaborated with me in the execution of the project that backed this thesis, namely Ricardo Silva, Pedro Daniel Sousa and Abel Santos. Also, to everyone with whom I've worked in the past few years, including, but not limited to, Ricardo Gimenes, Rodrigo Braga, Marcelo Petry and Pedro Abreu.

On a more personal note, I would also like to thank all my friends for all the support in the last years. A special note to my colleagues and friends at LIACC, and especially to Pedro Abreu, who was always by my side during this long ride.

Finally, and most importantly, I would like to thank my family, in particular my father, mother and grandmother, for all the support they have always shown me, and for putting up with my being 'absent' for so long. They are the safe port in the sea that is life, and for that, this thesis is entirely dedicated to them.

This work was supported by the Portuguese Foundation for Science and Technology (FCT), under Doctoral Grant with reference SFRH/BD/36610/2007.

Daniel Castro Silva

"When one door closes, another opens;
but we often look so long and so regretfully upon the closed door
that we do not see the one which has opened for us."

Alexander Graham Bell

Contents

List of Figures

List of Tables

Abbreviations and Acronyms

ACENET	Agent Collaborative Environment based on .NET
ACL	Agent Communication Language
ADE	Airport Design Editor
AFX	Airport Facilitator X
AI	Artificial Intelligence
AIIM	Association for Information and Image Management
AIP	Agent Interaction Protocol
AIXM	Aeronautical Information Exchange Model
AMS	Agent Management Service
ANGUS	Acoustically Navigated Geophysical Underwater System
ANSI	American National Standards Institute
AOSE	Agent Oriented Software Engineering
AP	Auto-Pilot
API	Application Programming Interface
APS	Aviation Performance Solutions
APX	Agent Programming eXtensions
ATC	Air Traffic Control
AUML	Agent UML
AUV	Autonomous Underwater Vehicle
BDI	Belief-Desire-Intention
CAPNET	Component Agent Platform based on .NET
CAST	Collaborative Agents for Simulating Teamwork
CATS	Crew Activity Tracking System
CCL	Common Control Language
CDL	Configuration Description Language
CLI	Common Language Infrastructure
CO	Carbon Monoxide
CO_2	Carbon Dioxide
COTS	Commercial Off-The-Shelf
CSV	Comma-Separated Values
DAI	Distributed Artificial Intelligence
DAML-S	DARPA Agent Markup Language for Services
DARPA	Defense Advanced Research Projects Agency
DATS	Durable Aviation Trainer Solutions
DDL	Disturbances Description Language
DF	Directory Facilitator
DOD	United States Department of Defense
EXCDS	Extended Computer Display System

FAA	Federal Aviation Administration
FCS	Flight Simulation Company
FCS	Future Combat Systems
FDM	Flight Dynamics Model
FEUP	Faculty of Engineering, University of Porto
FIPA	Foundation for Intelligent Physical Agents
FSX	Flight Simulator X
GIS	Geographical Information System
GML	Geography Markup Language
GPL	General Public License
GPS	Global Positioning System
GSR	Galvanic Skin Response
IATA	International Air Transport Association
ICAO	International Civil Aviation Organization
IED	Improvised Explosive Device
IMO	International Maritime Organization
IP	Internet Protocol
JADE	Java Agent DEvelopment Framework
JESS	Java Expert System Shell
KIF	Knowledge Interchange Format
KML	Keyhole Markup Language
KQML	Knowledge Query Manipulation Language
LEAP	Lightweight Extensible Agent Platform
LIACC	Artificial Intelligence and Computer Science Laboratory
LIDO	Laboratory of Informatics, Distributed Systems and Objects
MACE	Multi-Agent Computing Environment
MADKit	Multi-Agent Development Kit
MALLET	Multi-Agent Logic Language for Encoding Teamwork
MAS	Multi-Agent System
MaSE	Multiagent Systems Engineering
MDL	Mission Description Language
MTS	Message Transport Service
NADS	National Advanced Driving Simulator
NASA	National Aeronautics and Space Administration
NATO	North Atlantic Treaty Organization
NDB	Non-Directional Beacon
NRC	National Research Council
OGC	Open Geospatial Consortium
O-MASE	Organization-based MaSE
PDA	Personal Digital Assistant
PDT	Prometheus Design Tool
PID	Proportional-Integral-Derivative
ROADMAP	Role Oriented Analysis and Design for Multi-Agent Programming
ROV	Remotely Operated Vehicle
SAGE	Scalable Fault Tolerant Agent Grooming Environment
SAM	Surface-to-Air Missile
SCM	Supply Chain Management
SDK	Software Development Kit

SDL	Scenario Description Language
SPEM	Software Process Engineering Meta-Model
TAC	Trading Agent Competition
TACUND	Tower Air Traffic Control at the University of North Dakota
TAEMS	Task Analysis, Environment Modeling and Simulation
TCP	Transmission Control Protocol
TDL	Team Description Language
TFR	Temporary Flight Restriction
TMC	Toyota Motor Corporation
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
UGV	Unmanned Ground Vehicle
UK	United Kingdom
UML	Unified Modeling Language
US	United States
USARSim	Unified System for Automation and Robot Simulation
USCG	United States Coast Guard
UUV	Untethered Underwater Vehicle
VHF	Very High Frequency
VIN	Vehicle Identification Number
VMS	Vertical Motion Simulator
VOR	VHF Omnidirectional
W3C	World Wide Web Consortium
WASD	Wide Area Search and Destroy
XML	eXtensible Markup Language
XSD	XML Schema Definition

Chapter 1

Introduction

This chapter provides an overview of this thesis, and the developed work reported herein. First, the context and motivation are described, followed by a brief presentation of the main goals and possible applications of the developed platform and, also, a summary of the main contributions. Finally, the structure of this thesis is outlined.

1.1 Context and Motivation

In the past years, the number of operations and tasks performed by robots with different levels of autonomy has increased significantly. Some of the foremost examples involve the military applications of autonomous and remotely operated vehicles. The United States, for instance, has invested millions of dollars in autonomous vehicles and will continue to do so in the future [DOD, 2005].

The first applications used remotely operated vehicles (ROV) to perform tasks in environments that were either dangerous or inaccessible to human beings. Examples of such environments include mine fields [Das et al., 1999] or the bottom of the ocean (with applications both in the military [von Alt et al., 2001] and scientific areas [Ballard, 1993]), among others. One of the aspects that needs to be considered when using these vehicles is the interface used to maneuver the vehicle remotely [Fong & Thorpe, 2001].

One of the most visible application of semi-autonomous ground vehicles is the detection of improvised explosive devices (IED) [Miller, 2006]. Some robots have achieved some notoriety in this field, as is the case of TALON [GlobalSecurity.org, 2005b]. Another use of unmanned ground vehicles is to transport cargo, relieving soldiers from that task [GlobalSecurity.org, 2005a]. In order to promote research and advancements in autonomous ground vehicles, the Defense Advanced Research Projects Agency (DARPA) has promoted a challenge (the DARPA Grand Challenge[1]), consisting in off-road as well as urban scenario operations [Buehler et al., 2009]. This challenge and its tempting prize has contributed greatly for the technological advancements in unmanned

[1] More information available online from `http://archive.darpa.mil/grandchallenge/index.asp`

ground vehicles (UGV) [Seetharaman et al., 2006]. One of the most visible applications of UGV technology to everyday tasks is seen in several modern vehicles, which are capable of parking themselves with minimum or no aid from the driver [Vestri et al., 2005].

Underwater vehicles have also evolved over the past years. Initially powered and controlled remotely (named Remotely Operated Vehicles – ROV), they evolved in terms of autonomy, first in terms of power (Unmanned Untethered Vehicles – UUV) and then in terms of control and navigation as well (Autonomous Underwater Vehicles – AUV) [Blidberg, 2001]. These vehicles have practical applications in distinct fields, ranging from military [NRC, 2005] to scientific, including fields such as underwater archeology [Mindell & Bingham, 2001], and even commercial [Whitcomb, 2000]. More recently, autonomous surface water vehicles have also been the subject of research and attention [Cruz & Alves, 2008a].

Perhaps the most visible of autonomous vehicles are aircraft. Unmanned Aerial Vehicles (UAV) achieved a big notoriety due to their use by the US military during the last wars they participated in [Haulman, 2003]. Their commitment to the development and use of UAVs in the future is also visible in [DOD, 2005]. The use of UAVs is not limited to military scenarios, but several civilian UAVs have also been developed, probably the most notable ones used by NASA (National Air and Space Administration) [Nonami, 2007]. More recently, there has been a line of research that deals with UAVs of reduced size, which are more suited for some small-range missions or for urban environments [Hermans & Decuypere, 2005].

Some concerns still exist, however, when providing these vehicles with an increasing level of autonomy, and the human factor continues to be one important aspect when considering operations with autonomous vehicles [McCarley & Wickens, 2005]. Some form of manual override should always be present, allowing the vehicles to be remotely operated, and allowing some decisions the vehicle may have made to be revoked.

As can be seen, there are several areas where the number of operations with autonomous vehicles is increasing. As the level of autonomy of the vehicles increases, so does the desire to operate with multiple vehicles [Ryan et al., 2004]. These solutions allow only one operator to control multiple vehicles for the execution of a mission, thus increasing the overall performance of the system and decreasing the (human) requirements of the system.

1.2 Main Goals

The main goal of this dissertation is to design and implement a multi-agent system composed of a simulation environment and autonomous vehicles capable of self-coordination in order to accomplish high-level missions, such as forest surveillance and fire detection, target surveillance and tailing, providing aid in search & rescue operations, and other tasks, thus extending research on intelligent agent coordination.

Given the complexity involved in such goal, some more concise and measurable tasks have been defined:

- Development of a simulation platform that can be used in the implementation and evaluation of coordination and planning methodologies among heterogeneous autonomous vehicles. Such platform must include several components in order to accomplish that goal (an overview of such components can be found in section 3.2 and a detailed description of each component in chapters 5 and 6):
  - A comprehensive simulation platform that can be used to provide both a simulation engine capable of simulating autonomous vehicles in a realistic environment and a truthful visual feedback of the simulation;
  - A central application that can be used by an operator as an interface to the platform, allowing him to configure the platform and all the necessary elements for a mission to be performed;
  - A uniform manner in which to control different vehicles types, with a similar set of high-level maneuvers and commands;
  - A mechanism to detect and avoid possible conflicts (mostly collisions) between vehicles, effectively also providing with a centralized traffic control for the vehicles;
  - The necessary mechanisms for a simulation to be recorded for posterior analysis, and the methods to analyze such missions.
- Development of languages that can be used to provide a high-level description of all the entities necessary for the operation of such a platform:
  - Description of the scenario in which the simulation takes place, including all static entities present therein;
  - Description of the team of autonomous vehicles that will operate on the environment and their capabilities;
  - Description of the disturbances or abnormalities that exist in the environment, and their behavior and interaction with both environment and vehicles;
  - Description of the mission to be performed by the team.

1.3 Practical Applications

Several practical applications can be devised from this application, such as Search and Rescue operations, several military operations, civilian surveillance (fire, pollution), as well as some commercial applications. Some of these applications are described below.

Search & Rescue Search and Rescue can be defined as the cooperative use of resources in order to accomplish the goal of locating and rescuing a person or group of persons (or other material assets) that are either lost or in some way at risk.

In many situations, these operations are conducted either on a large area, or in an urban environment, where accessibility can be an important issue to consider. In such environments, the use of automated mechanisms becomes a requirement in order to decrease search times, and increase the probability of detecting and rescuing the person(s) in need in a timely manner [Casper & Murphy, 2003] [Goodrich et al., 2008].

In order to optimize the search, several patterns can be used, including track line, parallel, creeping line, sector, or square [USCG, 2006]. Each of these search patterns has its own advantages and disadvantages (such as the time spent covering the search area, the effective area covered by one sweep, or the detection probability, among others), and each pattern is better suited for particular situations [Frost, 1996].

Fire Detection One of the most visible applications is forest surveillance, in search for fires. This application can have a significant effect, since an early detection of the fire, before it spreads too much, can contribute to a faster response by the fire departments, thus avoiding damages to the local ecosystems or infrastructures. A fire changes the environment in several manners, and can be detected by sensing the changes it causes. These changes include temperature, concentration of several gases, such as carbon monoxide or carbon dioxide, visible smoke and flames, which can be detected using image processing software, and others. Many works have been published over the years regarding automatic fire detection using a combination of different sensors [Jackson & Robins, 1994], [Mueller & Fischer, 1995], [Pfister, 1997], [Gottuk et al., 2002]. Several research groups have envisioned the detection and monitoring of forest fires using Unmanned Aerial Vehicles to carry several sensors and ongoing research seems promising regarding this application [Casbeer et al., 2006], [Esposito et al., 2007], [Martínez-de-Dios et al., 2007].

Hydrothermal Vents Detection Hydrothermal vents are fissures on the planet's surface, from where geothermally heated water sprouts (on land, different types of vents exist, including fumaroles, geysers, hot springs and others – the most famous one probably being the Old Faithful geyser, located in Yellowstone National Park, in the northwest of continental United States [Rinehart, 1969]). Finding underwater hydrothermal vents (also known as sea vents or black smokers) or similar structures is a possible application of this platform.

The first underwater hydrothermal vents were discovered back in 1977 [Ballard, 1977], using the ANGUS (Acoustically Navigated Geophysical Underwater System) vehicle, a deep-towed two-ton vehicle, equipped with cameras and temperature sensors[2], and had a profound impact in many fields, such as geology, geochemistry, geophysics [van Andel & Ballard, 1979] and biology [Grassle et al., 1979] [Baross & Hoffman, 1985]. In 2000, a new set of vents was discovered in an unsuspected place, containing never-before seen structures that reach 60m in height, resembling underwater sky-scrappers (the area was named Lost City Hydrothermal Field) [Kelley et al., 2001]

[2]ANGUS was towed 2500 meters below the surface by the Research Vessel Knorr, using a steel cable. More information about the discovery of hydrothermal vents is available at `http://www.divediscover.whoi.edu/ventcd/`

[Kelley et al., 2005]. The minerals released by hydrothermal vents can also be of use to the industry, as underwater mining operations become a reality [Hoagland et al., 2010]. Some work has already been developed in order to automate the search for these vents, using autonomous underwater vehicles (AUV) [Yoerger et al., 2002] [Jakuba, 2007].

Pollution Detection and Measurement A similar problem is the detection of pollution and its source [Naden, 1971]. In the current context, where the concern with the environment becomes paramount, it is important to identify the major pollutants affecting the environment, and their sources, as to provide authorities with the information that allows them to take the necessary measures to stop or decrease the levels of pollution [Mashyanov et al., 2001], [Chavent et al., 2007], [Rajkovic et al., 2008], [Weimer et al., 2009].

A particular case of pollution detection is the detection of radiation. Solutions based on UAVs have already been proposed to provide with a means to perform atmospheric radiation monitoring and measurement [Stephens et al., 2000].

Mobile Target Detection and Pursuance A similar application, but considering a mobile target, consists in detecting a vehicle and following it after detection, as to determine its destination or behavior [Coifman et al., 2004], [Lee et al., 2003], [Rysdyk, 2006], [Rafi et al., 2006]. This type of mission is very important in a military or law-enforcement scenario, if the vehicle being targeted for tracking either belongs to the enemy forces, or is suspected of performing some illegal activity.

Illegal Activities Detection Another application is the use of infrared sensors to detect the cultivation of illegal drugs, such as marijuana [Bennett et al., 1996]. By using the appropriate sensors, the chemical products resultant of the production of certain drugs (such as methamfetamines) can also be detected by the vehicles. Another possibility is the detection of oil tankers performing illegal tank cleaning operations at sea.

1.4 Contributions

The contributions perceived to be the main contributions presented herein are:

1. Initial Contributions

    - **Generic Model** Associated with the developed platform, the model developed for generic robotic systems (described in section 3.1) is perceived to be an important contribution. It provides system designers with a starting point for their work, avoiding or shortening the common initial tasks involved in the design of systems comprised of several mobile (robotic) agents [Silva et al., 2011b].
    - **Use of Realistic Simulator** When exploring the capabilities of the chosen simulator, it was used as an immersive simulation environment that allowed for the assessment of the emotional state of a person (as described in section 5.1.2.2) [Silva et al., 2009b].

2. Developed Platform

   - **Platform** The developed platform, described in chapter 3, is one of the foremost contributions of this thesis. It allows for both a macro and micro vision of the simulation, through the use of a realistic simulator. It supports different vehicle types, including aircraft, ground vehicles, boats and submarines (the last not directly supported by the chosen simulator) [Gimenes et al., 2008], [Santos, 2010], [Sousa, 2010].
   - **Languages** The four developed languages and their classification and definition, described in chapter 4, are considered to be major contributions to the community, given that they can be used to specify, using high-level concepts, an operating scenario, team composition, disturbances in the scenario, and a mission for the team to perform [Silva et al., 2011d], [Silva et al., 2011c], [Silva et al., 2011a].

1.5 Document Structure

The remainder of this document is organized as follows:

- Chapter 2 presents a brief literature review on several topics related to the developed work – section 2.1 covering agents, multiagent systems and related topics; section 2.2 covering topics regarding simulation, with a focus on flight simulation; and section 2.3 covering some topics related to the developed languages.
- Chapter 3 presents a general model designed for systems with requirements similar to the one presented herein, followed by the specific platform architecture and its description, as well as an introduction to the four developed languages and their categorization.
- Chapter 4 describes the four developed languages in more detail, starting with a presenting of some implementation considerations, and followed by a complete description of the Scenario, Teams, Disturbances and Mission Description Languages.
- Chapters 5 and 6 are dedicated to the description of the main components of the developed platform. Chapter 5 focuses on the components the operator has some control over:
  - the Simulator, detailing its choice and some adaptations, as well as possible applications (section 5.1);
  - the Control Panel for the platform and its functionalities (section 5.2);
  - the Disturbances Manager, (section 5.3);
  - the Monitoring Tool, which allows the operator to be kept apprised of the status of a simulation session, and also allows for a more detailed monitoring of each of the vehicles (section 5.4)
  - the Logging and Performance Analysis tools, describing how simulation data is stored, and how the performance of the team can be evaluated when performing a mission (section 5.6)

Chapter 6 focuses on the autonomous agents that will populate the platform, providing with information regarding the chosen agent communication platform and describing in detail the following components:

  - the ATC agent, covering its details and operations (section 6.2);
  - the Vehicle Control Agent, covering several aspects related to vehicle control (section 6.3);

- Finally, chapter 7 presents an evaluation on several components of the platform, as well as the conclusions and some lines for future work.

Chapter 2

Literature Review

This chapter provides with a literature overview on some generic topics related to this thesis. It is divided into three large areas – agents and multi-agent systems; simulation environments, with an emphasis on flight simulation and related topics; and specification languages, that have some similarity in use or content with the developed languages. It is not the aim of this chapter to provide with a comprehensive and exhaustive literature review for the presented topics, but rather to provide an overview of several topics that are referenced throughout this dissertation.

2.1 Agents and Multi-Agent Systems

This section presents a brief overview of agents and multi-agent systems, as well as some agent-related topics. First, the concepts of agent is presented, followed by the characterization of the environment in which an agent exists, and the description of some generic agent architectures. Then, the concept and applications of multi-agent system are presented, followed by an introduction to coordination in multi-agent systems. An overview on agent communication platforms is presented in section 2.1.6, and some platforms are described. Finally, in section 2.1.7, an overview on agent-oriented software engineering methodologies is presented and some methodologies described.

2.1.1 Agents

Agent-oriented applications are more and more replacing traditional linear, monolithic or object-oriented ones. This phenomenon occurs mostly because agents are a good software paradigm to solve problems in open, heterogeneous and distributed environments, and because agents are able to solve problems that conventional centralized approached could not solve in a suitable manner.

In chapter 2.3 of [Wooldridge, 2002], Wooldridge sums the main differences between agents and objects to one slogan – "Objects do it for free; agents do it because they want to". This expression illustrates agents' ability to make autonomous decisions, thus maintaining control over

their own state and behavior. The notion of an agent is one that has developed from several scientific areas, from Artificial Intelligence (problem solving, logical reasoning, planning, learning, ...) to Software Engineering (agent-oriented programming, ...), and even Sociology (agent interaction, ...) or Game Theory and Economics (negotiation, conflict resolution, market mechanisms, ...), each of these areas contributing with a portion of typical agent behavior or capabilities [Reis, 2003].

There is no generally accepted global definition of agent, mostly due to the lack of a well defined programming paradigm for distributed systems, and the loosely manner in which the term is used to depict simple applications. However, based on several authors ([Smith et al., 1994], [Hayes-Roth, 1995], [Wooldridge & Jennings, 1995], [Maes, 1996], [Franklin & Graesser, 1996], [Russel & Norvig, 2002]), one can say that, in the artificial intelligence area, an agent is defined as a computational system with the ability to perceive the outer environment, through sensors, to decide and interact autonomously with that same environment, through the use of actuators, and possessing high-level communication capabilities, in order to perform the task it was designed to [Reis, 2003]. Figure 2.1 illustrates a typical schema of an agent, showing the interaction between the agent and the environment – the environment can be the real world (or part of it), when dealing with a robotic agent, or a simulated one, when dealing with software agents. Examples of robotic agent sensors include video cameras, microphones, infrared or ultra-sound proximity sensors, accelerations sensors, gyroscopes, temperature and humidity sensors, and many others. Actuators usually consist of engines and wheels, robotic arms and legs, speakers and others.

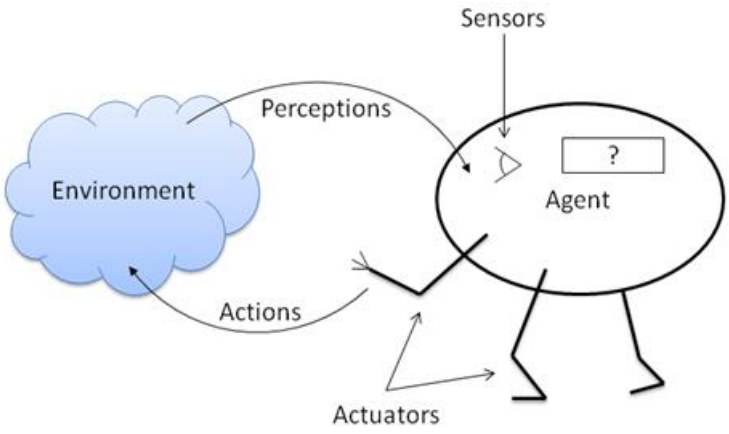


Figure 2.1: Generic Agent Schema (Adapted from [Russel & Norvig, 2002])

Several properties are imposed to an agent, given the above definition; others are desired in certain conditions or applications. Some of these properties are analyzed below.

- **Autonomy**. Although an essential and the most consensual characteristic in the definition of an agent, full autonomy cannot be attained, given that the agent needs to be created and activated by either a human being or another agent. However, according to Nwana, autonomy refers to the agents' capability to act based on an internal set of rules, without human guidance [Nwana, 1996].

- **Social Skills**. Related to agents' communications skills and interaction with other agents, social skills require the usage of a common high-level language and a shared ontology.

- **Reactivity**. This property relates to the agents' capability to rapidly react to changes in the environment, adjusting itself to it. The interest over reactivity led to a great deal of investigation on the area [Ferber & Drogoul, 1992]. Examples with ant colonies were used to prove that simple reactive agents can show collective traces of intelligent behavior, even solving tasks considered complex.

- **Pro-activity**. Also known as initiative, it represents independent behavior, actions being triggered not only by environmental changes but also according to the agent's goals.

- **Mobility**. Mobility can refer to spatial mobility of a robotic agent, or to the ability of a software agent to move across a computer network. This is particularly interesting when referring to information retrieval agents that work across the internet.

- **Cooperation**. Cooperation is the ability to work together with other agents, in order to achieve a common goal, or complete a common task. In order to cooperate, the agent should possess social skills, in order to communicate with other agents.

- **Learning**. The ability to learn allows an agent to improve its performance in a dynamic environment, by adjusting its behavior according to the success or failure of its previous actions.

2.1.2 Environment

The environment is the context where the agents operate, and where their sensors gather data from. Based on [Russel & Norvig, 2002], the environment can be categorized along six dimensions, which are described below:

- **Fully Observable vs. Partially Observable**. A fully observable environment is one in which an agent can access, through its sensors, all relevant environment state information for its decision making process. A partially observable environment, on the other hand, implies that part of the information is inaccessible, or inaccurate.

- **Deterministic vs. Stochastic**. A deterministic environment is one in which the result of an action, for a given environment state, is completely determined. In a stochastic environment, in contrast, the effects of an action are not completely determined, and are also affected by the presence of random variables or probabilities.

- **Episodic vs. Sequential**. In an episodic environment, the agent's actions can be divided into single, unrelated episodes, each one not affecting future ones. In a sequential environment, one choice can affect all future decisions.

- **Static vs. Dynamic**. An environment is static if it does not change during the time the agent is deciding the action to take. If the environment keeps changing while the agent is making its decision, the environment is said to be dynamic.
- **Discrete vs. Continuous**. This distinction can refer to environment state, perceptions, actions and how time is dealt with. Continuous implies a continuous range of values and discrete implies a finite number of possibilities.
- **Single Agent vs. Multi-agent**. Single-agent environments are those in which only one agent is operating. Multi-agent environments imply the existence of more than one agent. These can be either competitive (if another agents performance measure is opposite to the performance measure of the agent in question) or cooperative (if performance measures are similar).

According to the above definitions, a partially observable, stochastic, sequential, dynamic, continuous, multi-agent environment is the most difficult one to handle. The real world is a perfect example of such an environment, as is the one used in this dissertation (see section 5.1.1 for more information).

2.1.3 Agent Architectures

Some attempts of agent architecture classification have been made. In [Russel & Norvig, 2002], Russel and Norvig propose five different architectures:

- **Simple Reflex Agents**. Being the simplest kind, reflexive agents interpret the sensorial input and determine the proper action by means of matching the current state with previously existing condition-action rules, thus constituting purely reactive agents.
- **Model-based Reflex Agents**. This kind of agents improves the previous one by maintaining a representation of the world, which is updated dynamically with the perceptions. Consequently, the same perceptions in different moments of time will possibly generate different actions.
- **Goal-based Agents**. This type of agents also includes, alongside the environment model, information about the desired objectives, in order to determine the best course of action. Search and planning might be desired prior to making a decision.
- **Utility-based Agents**. Despite the proverb that says that all roads lead to Rome, some roads are better than others. This means that goals by themselves are not enough; a utility function is desired to measure the degree of goal satisfaction. In this scenario, actions are chosen in order to maximize the agent's utility function.
- **Learning Agents**. These agents are able to learn from experience, in order to improve their performance and thus become more capable than the original knowledge allows it to be. In order to learn, feedback about the effect of the agent's actions is necessary, and

when interpreted as penalty or reward, allows it to determine better actions to take, thus augmenting or replacing existing rules.

Earlier, in 1994, in [Wooldridge & Jennings, 1994], Wooldridge and Jennings proposed three architecture categories:

- **Deliberative Architectures**. These architectures follow the classical AI approach, where the agents possess an explicit symbolic model of the world, and decisions are made through logical reasoning.

- **Reactive Architectures**. These architectures do not include any kind of central symbolic world model, and do not use complex symbolic reasoning, attempting to make decision in real-time. The most well-known reactive architecture is Brooks' Subsumption Architecture [Brooks, 1986], which predicts the existence of several independent behaviors and a mechanism to select the best possible action at each moment, thus producing a behavior subordination hierarchy.

- **Hybrid Architectures**. As the name suggests, these architectures make use of the characteristics of both deliberative and reactive architectures.

The same authors (Wooldridge and Jennings) also presented, a year later, the concept of a layered architecture [Wooldridge & Jennings, 1995]. The several subsystems are organized in hierarchies that interact by levels. With horizontal layers, each layer is connected to sensorial input and produces an action suggestion, which may imply the use of a mediator, which chooses the appropriate action for each moment. With vertical layers, information flows from layer to layer, which implies that all layers must be working in order to produce a coherent result. Another architecture worth mentioning is the BDI (Belief-Desire-Intention) architecture [Rao & Georgeff, 1991]. It's an essentially deliberative architecture, where the agent's internal state is a set of mental states based on the notions of beliefs, desires and intentions. Beliefs represent information, thus describing the environment. Desires are the ultimate goals the agent must achieve and intentions correspond to a set of actions or tasks selected by the agent in order to achieve the desired goals. More complex architectures were also proposed, as is the case of the social agent architecture [Moulin & Chaib-draa, 1996]. These agents must possess explicit models of other agents, updating these models with information retrieved from perceptions and communications. Selecting the best architecture is a task that requires the consideration of the environment the agent will exist in, the tasks the agent will need to perform, the existence and characteristics of other agents in the environment, and so on. Although agent technology has many potential applications at the industrial and commercial level, the vast majority of these applications are actually multi-agent systems and not isolated agents (industrial applications, e-commerce solutions, entertainment and medical products, as well as scientific research competitions).

2.1.4 MultiAgent Systems

Multi-Agent Systems are systems composed by multiple agents, capable of both autonomous behavior, in order to satisfy their agenda, and interaction with other agents present in the system. A key concern is to manage interactions and activity dependencies between agents, or, in other words, to coordinate the agents [Lesser, 1999]. Although being a relatively recent area of study, it has registered an accentuated growth, with not only the appearance of international conferences, magazines and books about the subject, but also with the visibility it has achieved along with the social communication and general audience, with the RoboCup International competitions [RoboCup, 2010], [Kitano et al., 1997]. The emergence of Multi-Agent Systems is related to a number of motivations, including the need to provide more natural solutions for inherently distributed problems, or where knowledge or professionals were physically apart, the inadequacy of centralized programs to solve large problems, the need for faster, more robust and scalable applications, the need to maintain private part of the information and knowledge of the agents involved, the will to study individual and social intelligence and behavior, and many other [Reis, 2003], [Stone & Veloso, 2000]. Scientific research and practice in Multi-Agent Systems, which in the past has been called Distributed Artificial Intelligence (DAI), focuses on the development of computational principles and models for constructing, describing, implementing and analyzing the patterns of interaction and coordination in both large and small agent societies [Lesser, 1999]. DAI focuses on problems where several agents perform parts of a task and communicate in a high-level language. DAI can be divided into two main research areas [Bond & Gasser, 1988], [Durfee & Rosenschein, 1994]:

- **Distributed Problem Solving**. The problem at hand is divided into a number of smaller modules, or sub-problems, which are solved by separate entities (agents) that cooperate only on workload sharing and knowledge and result sharing.

- **Multi-Agent Systems**. The goal is to coordinate a number of autonomous agents, by coordinating knowledge, goals, skills and plans in order to collectively solve problems and perform tasks.

Coordination is divided into two distinct areas - cooperative coordination and competitive coordination. While the latter centers of attention on solving conflicts between self-interested agents (mainly through negotiation processes, where electronic markets and auctions take particular relevance), the former focuses on coordinating agents with a notion of common benefit (the most commonly used methodologies allow the definition of roles, definition of a structural organization, definition and allocation of tasks or multi-agent planning). Communication between agents requires a communications module, which can follow one of two major architectures [Huhns & Stephens, 1999] [Genesereth & Ketchpel, 1994]:

- **Direct Communication**. Each agent is responsible for handling all communication details, thus requiring other agents' communicational details also. A key problem with this architecture is possible system collapse, if all agents decide to send messages simultaneously.

- **Assisted Communication**. Partially solving problems caused by the above described architecture, agents delegate communicational details to facilitator agents, which are responsible for routing messages to their recipients. However, centralizing communication capabilities may introduce a bottleneck in the system. Also, these facilitator agents may have the ability to keep messages until they are successfully received, avoiding message misplace and guaranteeing message reception order - although this feature may introduce a slight degradation in communications time.

Huhns and Stephens define four kinds of agents, based on the two types of messages they can send and/or receive - assertions and questions [Huhns & Stephens, 1999]. In order to communicate, there is the necessity to develop a common language, specifying syntax, semantics, vocabulary, pragmatic and speech domain model. In the early nineties, two main developments were achieved by the Knowledge Sharing Effort [Genesereth et al., 1992], [Finin et al., 1994]:

- **KIF (Knowledge Interchange Format)**. KIF is meant to represent knowledge on a specific domain. It is based on first-order logic, using a prefix notation, making it possible to be interpreted by both humans and computer programs. It describes domain properties and relationships between objects, providing logical boolean operators, universal and existential quantifiers, and the most typical data types. It was primarily developed to express the contents of KQML messages.

- **KQML (Knowledge and Query Manipulation Language)**. KQML is a language for message-based communication between agents. It specifies all necessary information for message content understanding. Each message is comprised of a performative (message type), and a number of parameters with respective value. The natural growth of this language, however, has brought some problems, such as backwards compatibility issues, originated by modifications to the number and types of performatives [Labrou & Finin, 1997].

By the mid and late nineties, the Foundation for Intelligent Physical Agents (FIPA) started developing Multi-Agent Systems standards. One of these standards is the FIPA ACL (Agent Communication Language), a language similar to KQML, but containing a lot less performatives (only twenty, in comparison with the more than forty specified by KQML), and more adequate to negotiation processes [FIPA, 2002a]. In order to define a common and globally accepted vocabulary, with the same expressions and meanings, ontologies are typically used, specifying not only class taxonomy but also concepts, properties and relationships. Learning in Multi-Agent Systems is becoming an increasingly important area of study, not only because adjusting individual behavior to the group is important, but also because a better understanding of group or team learning is desired. Two types of multi-agent learning can be identified - interactive and individual [Weiß, 1996]. While the former relates to situations where agents collectively try to achieve their common learning goals, the latter pertains to situations where each agent tries to reach its own learning objectives, but the process is affected by other agents. In section 6.2 of [Sen & Weiß, 1999], Sen and Weiss describe the main categories for multi-agent learning characterization.

2.1.5 Coordination

The definition of coordination is not consensual among researchers, but from several different definitions [Malone & Crowston, 1994], [Jennings, 1996], one can say that coordination is the act of working together in a harmonious manner, in order to attain a common goal [Reis, 2003]. Coordinating agents is necessary or desired for several possible reasons - dependency relationships between agents (some researchers developed distinct classification methods for these dependencies [Wooldridge, 2002], [Malone & Crowston, 1994], [Sichman & Demazeau, 1995]), necessity to coordinate actions, either because no single agent possesses all knowledge or resources to solve the task at hand, or due to the existence of a set of global restrictions, such as cost, time, resources, that must be met [Jennings, 1996]. Also, efficiency concerns and anarchy and chaos prevention are seen as reasons for the utilization of agent coordination [Nwana et al., 1996]. Two major difficulties are identified in multi-agent coordination [Wooldridge, 2002]:

- Agents may be designed by different developers, with distinct goals, thus introducing the need to negotiate with other agents, in order to persuade them into cooperation.

- Given the agents' autonomy and real-time decision making, they must coordinate their actions with other agents present in the system also in real-time.

These difficulties, allied with the reasons mentioned above, led to the proposal of several coordination methodologies, some of which are briefly presented in the following pages. Two distinct approaches in Multi-Agent System construction are identified - systems comprised of competitive agents and those composed of cooperative agents. While the former usually involve many designers, and agents with their own agenda and motivation, interested in their own wellbeing, the latter are usually projected by the same person or team, and the agents have a notion of global utility and welfare. Competitive coordination will not be analysed in detail, in this work, since the current project entails cooperative coordination only. However, it is worth mentioning that negotiation is the most significant form of competitive coordination, with much research done in the area. Tambe makes a distinction between coordination and teamwork, the latter being a sub-area of the former [Tambe, 1997]. Teamwork requires that all elements of the team have the same common goals, cooperating amongst themselves to achieve those goals, while cooperation does not necessarily imply that common effort. Lesser and Corkill state that the goals of coordination are to assure that all parts of the problem are solver by at least one agent, assure that all agents interact in order to allow the tasks to be executed and integrated into the global solution, assure that all agents act in order to attain global goals in a consistent manner, and to assure that these goals are achieved within the computational and resource limits available [Lesser & Corkill, 1987]. A coordination model, as seen in Fig. 2.2, is a formal schema through which interactions between agents can be expressed. It handles creation, destruction, activities, communication, spatial distribution and mobility, as well as action synchronization, distribution and monitoring along time [Reis, 2003]. It can be seen as being comprised of three elements [Ciancarini, 1996]:

- **Coordination entities**. The entities to be coordinated. In a Multi-Agent Systems, these entities are the agents.
- **Coordination media**. These make communication between agents possible. A coordination medium can also aggregate agents that should be manipulated as a whole. They include physical interaction means, communication channels, and so on.
- **Laws of coordination**. These describe how agents coordinate themselves through the given coordination media and using a number of coordination primitives.

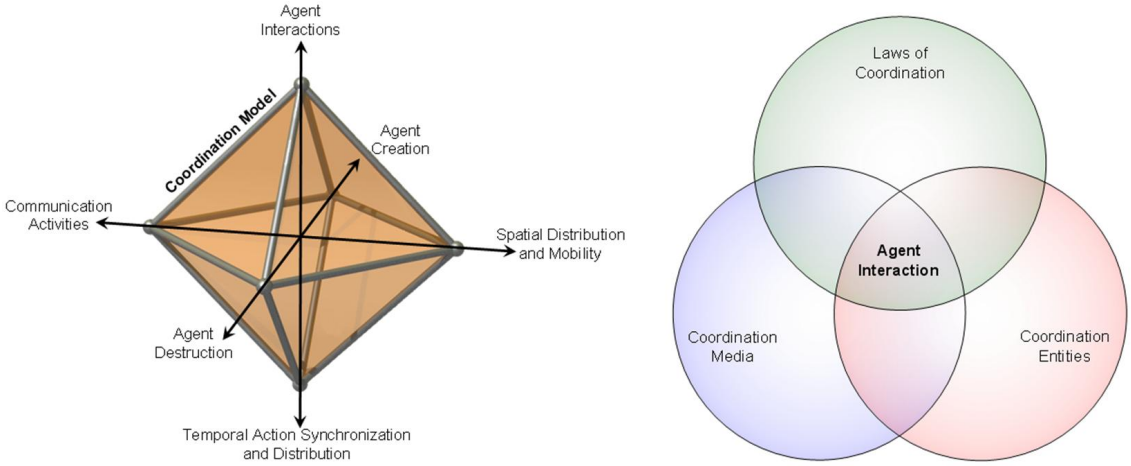


Figure 2.2: Coordination Model (Adapted from [Reis, 2003])

In order to measure the success of a given attained solution, Wooldridge proposed that coherence could be used, being measured in terms of quality, efficiency, resource usage, clarity, uncertainty and/or fault tolerance [Wooldridge, 1994]. A key identified problem is inconsistencies. These appear primarily due to the nature of the environment and the dimension of the system. To deal with inconsistencies, some methodologies were proposed, such as ignoring them, belief revision methodologies [Malheiro & Oliveira, 2000], treating them as conflicts and solve through negotiation, or build systems robust enough to reason even in the presence of inconsistencies [Lesser & Corkill, 1981]. One of the first coordination methodologies was suggested in the early eighties, by Smith and Davis [Smith & Davis, 1981]. They propose a three stage approach:

- Problem decomposition into smaller ones, usually in a hierarchical manner, with the least possible interdependencies.
- Individual solving of the small problems, which may involve task to agent allocation and information exchange during resolution.
- Solution integration, into the global solution, usually in the reverse order of the first stage problem decomposition.

This allows two major forms of cooperation – task-sharing and result-sharing. Regarding the task-sharing coordination, the Contract-Net protocol is the most popular method. The protocol starts

with the agent that needs a task to be done. It sends a message, specifying the tasks and restrictions, to agents capable of executing the task, which in turn respond with a proposal or refusal. The organizing agent then sends an acceptance or refusal of the received proposals [FIPA, 2002b]. Regarding result-sharing coordination, Durfee suggest that agents can improve their performance in terms of confidence, since combining all solutions, errors can be detected and confidence in the global solution increases; completeness, since combined local visions can produce a larger global vision of the problem and solution; precision, since shared results can improve global solution's precision; and time, since the necessary time to reach the solution can diminish, with the additional information received from other agents [Durfee, 1999]. Multi-Agent Planning is another type of coordination, with three identified possibilities - centralized planning of distributed plans, distributed planning of a global plan and distributed planning of distributed plans [Durfee, 1999]. Although a centralized planning is preferable, given the global problem vision held, it is not always possible, or desired, given the distributed nature of the problem and agents, or the need to maintain some information private. Partial Global Planning was proposed in the late eighties by Durfee and Lesser and is based on information exchange in order to reach a global solution for a problem [Durfee & Lesser, 1987]. It has three stages - goal generation, when each agent defines short-term plans to achieve its desired goals; information exchange, when agents exchange information about plans, goals and solutions; and local plan changing, when agent adjust their local plans according to the information they received. In order to avoid incoherence, a meta-structure called Partial Global Plan is used, containing the goals shared by all agents, activity maps, with each agent's activities and expected outcomes, and solution construction graph, specifying how agents should interact and exchange information in order to produce the global solution. Later on, in the mid-nineties, Decker extends this mechanism in his doctoral thesis, naming it Generalized Partial Global Planning [Decker, 1995]. He proposed five methodologies - updating non-local viewpoints, communicating results, handling simple redundancy, handling hard coordination relationships and handling soft coordination relationships. It also includes scheduling tasks with deadlines, heterogeneous agents and communication at multiple abstraction levels, making it much more flexible and practical. It was implemented in the TAEMS framework - Task Analysis, Environment Modeling and Simulation [Decker, 1996]. It allows the representation of complex agent coordination problems, as well as the analysis of the system's behavior, graphically showing tasks, agent actions and statistical data. It describes the environment and tasks in three levels - objective, subjective and generative. A problem is represented as a group of tasks, which is represented in a tree-like structure, where tasks are decomposed from the root to the leafs. The results are evaluated according to the necessary time to complete the tasks and the quality of the solution, measured by completeness, utility, precision, or any other desirable aspect. The Joint Intentions Framework was proposed in the early nineties [Cohen et al., 1990]. It is focused on the joint mental state of a team. It is every agent's responsibility to inform the remaining agents if the global objective has been attained, if it is impossible to attain the global objective or if there is no valid reason to attain the global objective. In the late nineties, Stone and Veloso introduced the Locker Room Agreement [Stone & Veloso, 1998], [Stone, 2000]. It is a high-level coordination mechanism useful in

reduced communication domains. It is based on a team flexible structure definition, based on roles, specifying agent behavior and role switch mechanisms; formations, composed of a set of roles and trigger conditions for their activation; and set-plays, for execution in specified conditions. Despite some flexibility introduced by the redefinition of the notion of formation, it has some limitations - for systems which include spatial location, individual roles have to be defined for each agent, which in some scenarios can be extremely elevated. In the mid-eighties, mutual modeling was proposed by Genesereth [Genesereth et al., 1986]. According to this approach, each agent creates a model of every other agent in the team, thus allowing it to predict their actions. A similar cooperation method was used in MACE, one of the first testing environments for Multi-Agent Systems [Gasser et al., 1987]. Based mainly on [Balch & Arkin, 1994], an intelligent perception method was proposed [Reis & Lau, 2001b]. By making intelligent use of sensors, it is possible to monitor other agents' actions in the environment, thus facilitating coordination and cooperation, without communication. Several more methodologies were proposed, many of them applied to the RoboCup environment [RoboCup, 2010], [Lau & Reis, 2007], [Mota et al., 2011]. However, two problems are identified - most implemented Multi-Agent Systems do not provide the flexibility to dynamically rearrange teams and roles according to necessities in their coordination mechanisms, and on the other hand, most coordination methodologies do not deal well with spatial mobility, and therefore are not applicable to robotic agent teams operating in the real world, or realistic simulations of it.

2.1.6 Agent Communication Platforms

Agent communication platforms are platforms that provide support for the development of agent-based applications, namely by abstracting developers from the specifics of agent-communication implementation. Since the standards for agent communication appeared, several platforms have been developed and proposed by the scientific community, implementing agent communication using these standards and freeing developers from having to implement them.

These platforms usually feature a number of services, as specified by the FIPA Agent Management Reference Model [FIPA, 2004]. The most basic services are the Message Transport Service (MTS) and the Agent Management Service (AMS). The MTS provides an infrastructure that guarantees that messages sent from one agent to another are delivered, independent of the receiving agent being on the same platform or on another one. The AMS, also called the white pages service, is a central registration service, gathering information regarding all agents present in the platform, which can be consulted by other agents. Another desirable service is the Directory Facilitator (DF), which, even though not mandatory, is usually present in most platforms. The DF service, also called yellow pages service, is used by agents to register the services they offer to other agents and to query for services offered. Additional features such as agent mobility, load balancing, persistence, logging and others are also among the desirable features of the agent platform to choose.

Some known agent-communication platforms are presented below, as to provide an overview of existing platforms and their characteristics.

In [Nguyen et al., 2002], a comparison of several agent communication platforms is presented. In [Leszczyna, 2008], the author also provides a comparison between several platforms, focusing the analysis on the currentness and popularity of the platforms.

2.1.6.1 FIPA-OS

One of the first platforms to be developed with the FIPA Reference Model in mind was the FIPA-OS[1], which was initially released in August 1999 as the first publicly available implementation of FIPA technology. With the purpose of providing a 'reference implementation' of the FIPA open standard for agent interoperability, the FIPA-OS is distributed as freely available and modifiable source code, entirely implemented in Java. It enables the adoption of FIPA standards without the need to implement the specifications, and at the same time assists in validating and evolving FIPA standards[Poslad et al., 2000]. FIPA-OS includes an extension (JESS Agent Shell) that provides support for writing FIPA-OS Agent which can take advantage of the JESS[2] system[Friedman-Hill, 2003]. The latest available version of FIPA-OS, however, was released in March 2003, and for the past six years no developments to the platform were registered.

2.1.6.2 JADE

One of the most widely known platforms complying with FIPA specifications is JADE (acronym for Java Agent DEvelopment Framework)[3], an open source platform for peer-to-peer agent-based applications, implemented in Java and distributed by Telecom Italia [Bellifemine et al., 1999]. JADE provides a number of services, including integration with JESS, the possibility to run on mobile platforms (by using LEAP[4] libraries), and several utilities that support the debugging phase, usually more complex in distributed systems [Bellifemine et al., 2007]. JADE is continuously being improved and new versions are released periodically. Jad-lite, Jini

2.1.6.3 Zeus

Another known tool is Zeus[5], an open source agent development toolkit, implemented in Java, that provides facilities to implement BDI-style agents, agents with reactive rule bases, agents with intelligent message handling functionality and DAML-S[6] service descriptions [Nwana et al., 1998]. Besides providing the common services, Zeus also provides a coordination engine, which includes a planner and scheduler that allows for the planning and execution of tasks that require coordination [Collins et al., 2000], as well as a methodology for the development of agents using the Zeus toolkit [Collis & Ndumu, 2000]. Despite being around for over a decade now, the latest update to

[1] More information available at `http://fipa-os.sourceforge.net/index.htm`
[2] More information available at `http://jessrules.com/`
[3] More information available at `http://jade.tilab.com/`
[4] Lightweight Extensible Agent Platform
[5] More information available from `http://labs.bt.com/projects/agents/zeus/`
[6] DARPA Agent Markup Language - Services. See `http://www.daml.org/services/owl-s/` for more information

Zeus, however, was released early in 2006, with no new announced developments in the past three years.

2.1.6.4 SAGE

SAGE (acronym for Scalable Fault Tolerant Agent Grooming Environment)[7] was built with the intent of providing a distributed decentralized, fault tolerant, scalable and lightweight agent platform that complies with FIPA specifications [Ghafoor et al., 2004]. Developed with the collaboration of the Institute of Information Technology (now School of Electrical Engineering and Computer Science) of the Pakistani National University of Sciences and Technology (NUST-SEECS) and Comtec Japan, some versions have been released over the past few years, including a light-weight version (SAGE Lite, which features light-weight versions of the AMS, DF, MTS and ACL modules) [Khalique et al., 2007]. SAGE's decentralized communication architecture allows for scalable, fault tolerant applications.

2.1.6.5 MadKit

MadKit (Multi-Agent Development Kit)[8] is a modular and scalable multiagent platform written in Java and built upon the AGR (Agent/Group/Role) organizational model [Gutknecht et al., 2000], [Gutknecht & Ferber, 2001]. Agents in MadKit may be programmed in Java, Scheme (Kawa), Jess or BeanShell, and other script language may be easily added. With around a decade of life, MadKit continues to be developed by the Montpellier Laboratory of Informatics, Robotics, and Microelectronics, and features such as agent mobility are currently being developed for future versions of the platform.

2.1.6.6 JACK

JACK[9] is a lightweight, cross-platform environment for building, running and integrating multi-agent systems, developed by Agent Oriented Software (AOS) [AOS, 2005a]. AOS provides some extensions to JACK, including JACKTeams (which allows for the definition of teams and their roles, capabilities, beliefs and knowledge [AOS, 2005b]), a BDI (Beliefs, Desires, Intentions) reasoning model and CoJACK – a cognitive architecture used for modeling variations in human behavior. Being a commercial product, new and improved versions of JACK are likely to continue being developed in the future.

2.1.6.7 CAPNET and ACENET

CAPNET (Component Agent Platform based on .NET) is a platform introduced in 2004 targeting the .Net framework [Contreras et al., 2004a] [Contreras et al., 2004b]. ACENET (Agent Collaborative Environment based on .NET) appeared later as a decentralized and fault-tolerant platform

[7] More information available online at `http://sage.niit.edu.pk/`
[8] More information available online at `http://www.madkit.org/`
[9] More information available online at `http://www.agent-software.com.au/`

[Mallah et al., 2009] [Ali et al., 2010]. Both CAPNET and ACENET are based on the .Net framework, unlike the other existing platforms, which are for the most part based on the Java programming language. However, no additional information on either of these projects is found, suggesting that they may have been abandoned.

2.1.6.8 AgentService

AgentService[10] is an agent platform based on the Common Language Infrastructure (CLI) and the *C♯* language [Grosso et al., 2003]. AgentService, together with APX (Agent Programming eXtensions, a set of extensions to the *C♯* language that simplifies the development of agents targeting the AgentService framework [Vecchiola et al., 2003]), offers a complete support to the development of multi-agent systems [Boccalatte et al., 2004]. It is developed by the Laboratory of Informatics of the Department of Communication Computer and System Sciences of the University of Genoa, in cooperation with Siemens, and [Vecchiola et al., 2008].

2.1.7 Agent Oriented Software Engineering Methodologies

The ever-growing use of agent-based technology and applications has brought forward the necessity to develop methodologies that could aid designers not only in development and deployment, but also in the early analysis and design phases of a project, in a manner similar to what traditional software engineering techniques have done for more conventional software projects, namely when using an object-oriented paradigm [Iglesias et al., 1999].

In this section, some of the most widely known AOSE methodologies that have emerged in the past years are briefly introduced (for a more complete and detailed review of existing methodologies, see [Bergenti et al., 2004] or chapter 7 of [Sterling & Taveter, 2009]).

Several publications can be found comparing some of these methodologies and other existing ones – see [Iglesias et al., 1999], [Bayer & Svantesson, 2001] (a detailed analysis of the Gaia and the MAS-CommonKADS methodologies), [Dam & Winikoff, 2003] (the authors present an in depth analysis of MaSE, Prometheus and Tropos) or [Sturm & Shehory, 2004] (the authors present a comprehensive comparison between Gaia, Tropos, MaSE according to several features, grouped into categories), among others.

As agent-oriented methodologies continue to be developed, research will keep aiming at the direction of determining which agent-oriented methodologies are best suited to support the development of a particular project or system.

The Gaia methodology uses an organizational view to construct MAS. Gaia has been recognized as a valuable methodology for the development of open complex systems based on the multi-agent approach but in order to be used in the development of real world systems, it needs to be extended in several aspects [Gonzalez-Palacios & Luck, 2008].

[10] More information available at `http://www.agentservice.it/`

2.1.7.1 AUML

AUML is an extension of UML (Unified Modeling Language) for agents [FIPA, 1999]. The most commonly used extension is the interaction model, in which each entity is seen as an independent agent [Odell et al., 2001] [Bauer et al., 2001]. Other UML models can also be adapted to represent agent-like features.

2.1.7.2 Tropos

Tropos was introduced in 2002 as a comprehensive AOSE methodology, encompassing all stages of project design, from early requirements elicitation to detailed design [Bresciani et al., 2002] [Giunchiglia et al., 2003]. Tropos can be considered as loosely based on a use-case model used in traditional Software Engineering methodologies. Its key concepts include actor, goal, plan, resource, dependency, capability and belief [Bresciani et al., 2004]. During the early requirements elicitation, actors and goals are identified from stakeholders and their objectives, using a goal-oriented analysis. Dependencies between actors and goals are also identified. In the late requirements stage, all functional and non-functional requirements for the system are specified in more detail. In this stage, the system is considered as a single actor, while external entities present in the environment are considered as interacting actors. The architectural design stage produces a model of the system architecture, describing how components work together. During the detailed design stage, detailed models of each component are produced, showing how goals are fulfilled by agents. In this stage, details such as agent communication language and protocols are specified using a more detailed modeling language such as UML [Giorgini et al., 2004].

2.1.7.3 Prometheus

Prometheus is methodology introduced in 2002 as a result of industry and academy experience [Padgham & Winikoff, 2003]. It provides support and a detailed process for specification to implementation stages of a project, and includes concepts such as goals, beliefs, plans and events. The methodology includes three phases: the system specification phase, which focuses on the system as a whole, identifying goals, functionalities and use case scenarios with the environment; the architectural design phase, which uses the models produced in the previous stage to determine the agents that will be present in the system, how they will interact with each other and react to events in the environment; and the detailed design phase, which produces detailed diagrams of each agent's functionalities and capabilities, as well as several other implementational details [Winikoff & Padgham, 2004]. A tool named PDT (Prometheus Design Tool) was developed to provide support to the Prometheus methodology in the design of agent systems [Padgham et al., 2008]. Many Promehteus concepts also map directly into the JACK system[11], which can be used to generate agent skeleton code [Padgham & Winikoff, 2004].

[11] More information about the JACK system and more recent developments available online at `http://www.agent-software.com.au/products/jack/index.html`

2.1.7.4 MaSE

The Multiagent Systems Engineering (MaSE) methodology was introduced in 2000 as a comprehensive methodology, including analysis and design stages [Wood & DeLoach, 2001]. It uses a number of graphical models to describe goals, behaviors, agent types, and agent communication interfaces, also providing detailed definition of internal agent design [DeLoach et al., 2001]. In the first analysis phase, goals are determined and structured by analyzing an already existing initial system specification. The second analysis phase is centered around use cases, detecting roles, use cases and use case scenarios from the system specification. In the third analysis phase, the identified roles are refined, producing a more detailed description of each role and their respective goals, and interactions with other roles. In the design stage, roles are mapped into specific agent classes; communication protocols between agent classes are detailed; the internal details of each agent class are defined, using components and connectors; and finally a system-wide deployment diagram is created. A tool named agentTool was developed to support the MaSE methodology (and more recently the Organization-based MaSE, or O-MaSE), from the initial system specification to implementation, using a set of inter-related graphical models [DeLoach & Wood, 2001].

2.1.7.5 Gaia

The original Gaia methodology was proposed in 2000, by Wooldridge, Jennings and Kinny, entailing both analysis and design phases, but not requirements elicitation or implementation [Wooldridge et al., 2000]. In the analysis stage, a roles model (containing the key roles in the system, their permissions and responsibilities, along with the protocols and activities in which they participate) and an interaction model (containing the patterns of inter-role interaction) are produced. In the design stage, an agent model (aggregating roles into agent types), a services model (derived from the activities and protocols of each role) and an acquaintance model (defining communication links between agent types) are produced. In the next paragraphs, we will analyze some proposed extensions to the Gaia methodology (a more detailed analysis of some of these extensions can be found in [Cernuzzi et al., 2004]).

The official extensions to Gaia (referred to as Gaia v.2 as to avoid ambiguity) were introduced in 2003, by Zambonelli, Jennings and Wooldridge, enriching the original methodology [Zambonelli et al., 2003]. The analysis stage was expanded to include an organizational model (decomposing the system into sub-organizations), an environment model (describing the environment in which the MAS will be situated) and the organizational rules (containing global organizational rules the system must respect and enforce). The design stage was divided into architectural design and detailed design stages. In the first, the roles and interaction diagrams are completed, and an organizational structure model is introduced (containing the structure, topology and control regime of the system). In the second, the agent and service models are created (as in the original methodology). Figure 2.3 shows these models and their relations in the Gaia v.2 methodology in more detail.

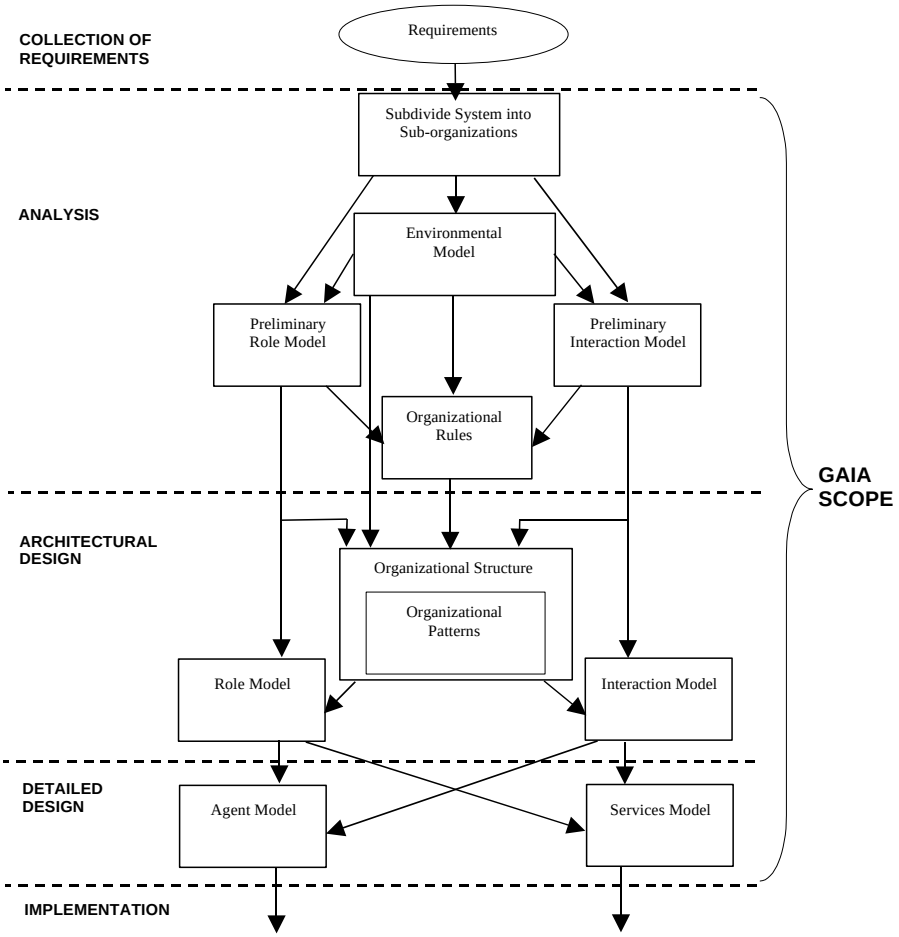


Figure 2.3: Models in the Gaia v.2 Methodology ([Zambonelli et al., 2003])

Some other extensions have been proposed by several authors. Some of these extensions include:

- The ROADMAP (Role Oriented Analysis and Design for Multi-Agent Programming) extensions to Gaia were proposed in 2002 [Juan et al., 2002] and later further extended[12]. ROADMAP introduces new features to the Gaia methodology, in order to eliminate or mitigate some identified weaknesses: support for requirements gathering (by introducing a use-case model); new models to describe the domain knowledge and the environment (knowledge and environment models, respectively); levels of abstraction that allow iterative decomposition of the system; models and representations of social aspects and individual characteristics; and runtime reflection modeling, to allow changes of social and individual aspects in runtime – by allowing roles to have read, write and change permissions on roles, role attributes (such as protocols) or a member of an attribute (a specific protocol, for instance).

[12]More publications about the ROADMAP methodology are available online from `http://www.agentlab.unimelb.edu.au/publications/Keyword/ROADMAP.html`

- Agent UML (AUML[13]) was introduced in the year 2000 as a set of UML idioms and extensions for dealing with agents [Odell et al., 2001] [Bauer et al., 2001]. In 2004, Cernuzzi and Zambonelli propose that the Agent Interaction Protocol (AIP) (the core part of AUML) be used in conjunction with Gaia, as to provide a richer, more compact and formal notation for agent interaction, reducing ambiguity and allowing the specification of multiple lifelines for the agent to choose from [Cernuzzi & Zambonelli, 2004]. Although this had already been suggested earlier – for instance, in [Juan et al., 2002](page 6) or [Zambonelli et al., 2003](page 348) –, it had never been detailed.

- Palacios In [Gonzalez-Palacios & Luck, 2008], Gonzalez-Palacios and Luck extended the Gaia methodology by introducing an agent design phase, and enhancing the methodological process with the use of iterations. The agent design phase follows the detailed design phase of Gaia and produces an object-based specification from which an implementation can be derived. This approach does not depend on a specific agent architecture, and it allows developers to select the architecture that best models a given agent. The use of iterations provides Gaia with a flexible methodological process that facilitates the development of large systems, by the reason of decomposing the development into iterations.

- Castro In [Castro & Oliveira, 2008], Castro and Oliveira used the Gaia methodology for modeling an airline company operations control center, and propose some complements (and replacements) to some of its models. They propose the replacement of protocol tables with UML 2.0 interaction diagrams; the formal notation of the organizational structure with UML 2.0 diagram; the agent model with a UML 2.0 class diagram; and the service model with a UML 2.0 class diagram. They also suggest to jointly use a UML 2.0 representation of roles and interaction diagram to help to better visualize roles, activities and protocols; and a few combined graphical representations to complement the preliminary role and interaction models and the organizational structure.

2.2 Simulation Environments and Flight Simulation

This section first presents a brief overview on simulation environments in general, then focusing on flight simulators in particular, and introducing some generic aspects related to flight, such as auto-pilot systems.

2.2.1 Simulation and Simulation Environments

In general terms, simulation can be defined as the imitation of some real thing, state of affairs or process [Rosen, 2008]. This somewhat vague definition usually entails a system that simulates reality, or part of it. Initial simulation methods relied heavily on the Monte Carlo method (run a large number of random simulations and observe the results [Mooney, 1997]) for obtaining results

[13] More information available online at `http://www.auml.org/`

but an increasingly computational component has been introduced that allows for a closer to reality simulation. A brief overview on the history of simulation can be found at [Nance & Sargent, 2002] or [Goldsman et al., 2009].

Simulation is an important tool in research, for testing and validating concepts, when they cannot be determined mathematically or in the real world, either because that would be too costly, time-consuming or even disruptive (simulation, however, may not be appropriate if the cost or time it takes to build the simulation system exceeds the cost of real operations).

Simulation now has a wide range of different application fields, from entertainment (movies [Swartout et al., 2005], games) to education (as training tools) and research, with military (training, war games [Macedonia, 2002]), healthcare [Brown et al., 2001], industrial (aerodynamics, process optimization, prediction (what-if scenarios), decision support systems, transportation and logistics [Rozinat et al., 2009]) or robotics (tool design and test [Dàvila-Rìos et al., 2008]) applications, among others [Johnson, 2008].

One application of simulation focusing on economy is the Trading Agent Competition (TAC), comprised of several challenges. One such challenge is the Supply Chain Management (SCM)[14], which is designed to promote the development of software agents capable of managing part of a supply chain [Arunachalam & Sadeh, 2005] [Collins et al., 2006]. Agents developed for this competition can be designed to win [Vinhas et al., 2007], or to test a given concept in a simulated environment [Abreu et al., 2007].

Several simulation environments emerged over the years, some with generic purposes, others with more specific applications. Among some generic robotic simulators are USARSim, Microsoft Robotics Studio and Player.

USARSim was designed as a high fidelity simulation of urban search and rescue robots and environments intended as a research tool for the study of human-robot interaction and multi-robot coordination, and was built over the Unreal Engine[15] [Wang & Balakirsky, 2008]. This simulator is used in the RoboCup Rescue Virtual league[16] [Balakirsky et al., 2007] [Balakirsky et al., 2006]. Figure 2.4(a) shows the simulation of an urban disaster scene using USARSim.

Microsoft Robotics Studio[17] was released by Microsoft in an attempt to create a software standard for robot development and control (and introducing Windows and the .Net framework to an area that traditionally used Linux-based tools) [Jackson, 2007]. It can be used for the rapid development and deployment of simple robotic simulations [Rango & Nahavandi, 2007] [Morgan, 2008]. Figure 2.4(b) shows a simulation using Microsoft Robotics Studio.

Another popular simulator is Player, a free-software platform for robotics simulation[18]. It includes a 2D interface names Stage and also a 3D simulator named Gazebo, and is very widely used by researchers and academics [Gerkey et al., 2003] [Rusu et al., 2007].

[14] More information available online at `http://www.sics.se/tac/page.php?id=13`
[15] More information about the Unreal Engine available from `http://www.unrealtechnology.com/`
[16] More information about the RoboCup Rescue available at `http://www.robocuprescue.org/`
[17] More information available from `http://www.microsoft.com/robotics/`
[18] More information available from `http://playerstage.sourceforge.net/`

(a) USARSim (b) Microsoft Robotics Studio

Figure 2.4: USARSim and Microsoft Robotics Studio

2.2.2 Fligh Simulation

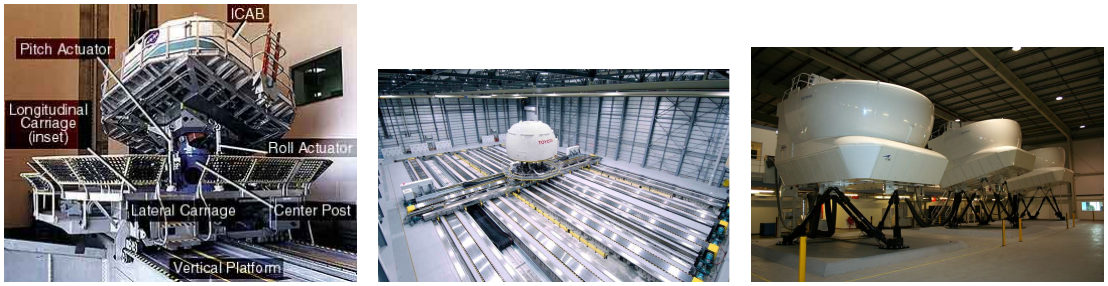
With one century of history [Page, 2000], flight simulators appear in a wide range of configurations, diverging in complexity, physical and/or simulated nature or orientation to professional training or entertainment (games). Early flight simulators consisted of human-operated constructions that simulated the effects of the aircraft controls in the body of the aircraft. These simulators have evolved greatly over the years, and nowadays physical simulators (or full flight simulators) are usually mounted on motion platforms that occupy large building and are capable of six degrees of freedom. One example of such platform is known as the Stewart platform [Stewart, 1966]; this platform is powered by six jacks, and allows for motion in all linear and rotation axis.

Examples of full flight simulators include the NASA VMS (Vertical Motion Simulator), a full simulator capable of six degrees-of-freedom movements, housed in a ten-story building in Ames Research Center, California [Danek, 1993] [NASA, 2008]. This simulator is capable of providing different simulation experiences, thanks to the ICAB (Interchangeable Cab) feature, which allows for a modular interior of the simulation cabin, presenting the pilot with different cockpit interfaces – see Fig. 2.5(a). Approximately 1300 variables can be retrieved from the simulator, allowing for thorough data analysis of simulation sessions. This simulator has been used for pilot and astronaut training but also in cooperation with other entities [Tran & Hernandez, 2004]. Other examples of full simulators, but targeting ground vehicles, can also be found. One such example is the Toyota Driving Simulator, a full simulator developed by the Toyota Motor Company that uses a real car placed inside a 7-meter dome with a 360-degree concave video screen – see Fig. 2.5(b) – to analyze the driving characteristics of average drivers (and their reactions to specific situations) and to aid in the development and verification of active safety technology for reducing traffic accidents [Toyota Motor Corporation (TMC), 2007]. Another example is the National Advanced Driving Simulator (NADS), developed by the National Highway Traffic Safety Administration[19] and located at the University of Iowa [Stall & Bourne, 1996] [Schwarz et al., 2003]

[19]More information available from `http://www.nhtsa.gov/Driver-Simulation`

[Ranney et al., 2003]. These full simulators, however, do not exist in great numbers, mainly due to their cost (NADS is a 50 million dollar project), and also the required physical space (large multi-story warehouses).

Smaller simulators are usually used for pilot training purposes in a stationary basis, allowing for a better use of space; for instance, flight training companies, such as the Flight Simulation Company (FSC[20]) (shown in Fig. 2.5(c)), SimCom Training Centers[21] or APS[22] provide full flight simulator facilities for pilot training in a number of different aircraft types.



(a) VMS Diagram (b) Toyota Driving Simulator (c) Full Flight Simulators

Figure 2.5: Diagram of the NASA VMS, Dome and Building of the Toyota Driving Simulator and a Row of Full Flight Simulators at FSC

The evolution of simulators is closely tied to the evolution of computational capabilities. Early simulators used hardware and software purposely built for that effect. Nowadays, virtually any computer is capable of running flight simulation software. Some simpler flight simulators (as is the case of earlier simulators, or light-weight flight simulation games) simulate only the four basic forces of flight – weight, lift, thrust and drag – see Fig. 2.6. Simulators using only these four forces, however, allow only for a simple simulation that does not consider the effects of changes in the environment surrounding the aircraft and the complexities of the interaction between aircraft and environment.

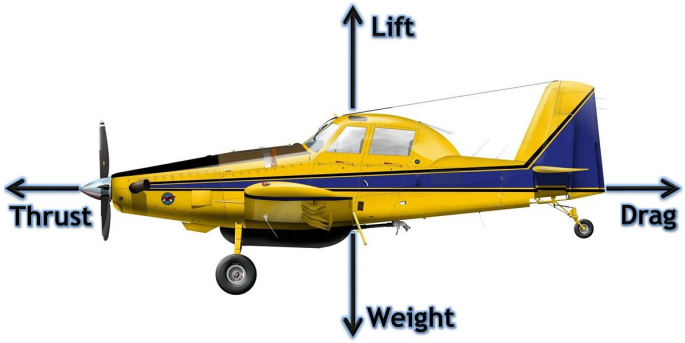


Figure 2.6: Simplified Forces of Flight

[20] More information available online at `http://www.fsctraining.com/`
[21] More information available online at `http://www.simulator.com/`
[22] More information available online at `http://www.apstraining.com/`

Other more realistic and complex simulators (usually professional simulators used in the industry) take many things into account, such as the complexities of the weather, the design of the aircraft itself, differences in propulsion systems, and many other factors.

The advancements of recent years have bridged the gap between dedicated professional simulation software and that aimed at entertainment purposes (game engines). In fact, some existing game engines are used by professionals and for research [Lewis & Jacobson, 2002]. Among these game engines, a few flight simulators stand out (these simulators are analyzed in more detail in section 5.1.1):

- **FlightGear.** FlightGear is a free, open-source, multi-platform, cooperative flight simulator, and the source code for the entire project is available and licensed under the GNU General Public License[23]. The goal of the FlightGear project is to create a sophisticated flight simulator framework for use in research or academic environments [Perry, 2004] [Sorton & Hammaker, 2005]. Some of the most publicized features of FlightGear include the flexibility of choosing a Flight Dynamics Model (FDM) among the existing ones, or adding new ones (see section 5.1.1.2 for more information about the FDMs included in FlightGear), extensive and accurate world scenery and sky model, flexibility of aircraft modeling and data access and communication with external modules. Figure 2.7(a) shows a screenshot of the FlightGear flight simulator.

- **X-Plane.** X-Plane is a well-known multi-platform, very ambitious flight simulator project, by Laminar Research, with a growing number of fans. X-Plane is known for the dimension of the scenario visual data (about 70 Gigabytes), and for the use of a geometric approach to simulation, which allows it to be used for testing the design and aerodynamic efficiency of new aircraft, and includes subsonic and supersonic flight dynamics, and support for flying wings and fly-by-wire systems. One aspect that stands out about X-Plane is the fact that it has been approved for a very wide range of FAA[24]-certification levels with a wide range of companies (in conjunction with the necessary simulation hardware). It has also been used by several research groups as a simulation basis [Garcia & Barnes, 2010], [Ribeiro & Oliveira, 2010]. Figure 2.7(b) shows a screenshot of X-Plane.

- **Flight Simulator X.** FSX, short for Flight Simulator X, is the tenth version of the acclaimed Microsoft Flight Simulator series, perhaps one of the most known simulation environments in the gaming community[25]. Having evolved greatly since the first release thirty years ago [Grupping, 2008], Flight Simulator now presents an improved realism, featuring a rich scenario, with independent vehicles (including airport ground vehicles, cars

[23] See `http://www.flightgear.org/Downloads/source.shtml`

[24] Federal Aviation Administration, an agency of the United States Department of Transportation with authority to regulate and oversee all aspects of civil aviation in the United States

[25] More information regarding the Flight Simulator series can be found at `http://fshistory.simflight.com/`

moving in highways, boats and ships in lakes and oceans, and air traffic) as well as animal herds, a new mission system that allows for the definition and execution of a number of different missions, and many other improved features. FSX is also used as a tool by several researchers [Cantoni & Neto, 2008] [de Farias et al., 2007] [Kenny et al., 2008] [Diehl et al., 2009]. Figure 2.7(c) shows a screenshot of FSX.

(a) FlightGear

(b) X-Plane

(c) FSX

Figure 2.7: FlightGear, X-Plane and FSX

2.2.3 Auto-Pilot Systems

An auto-pilot system (or simply autopilot) is a mechanical, electrical, or hydraulic system used to guide a vehicle without assistance from a human being. Although auto-pilot systems are more commonly associated with aircraft, they can be used in any kind of vehicle, including boats, cars, or even missiles [Pellanda et al., 2002] [Faruqi & Vu, 2002].

Autopilot systems applied to boats and ships usually contain steering, course keeping or maneuvering capabilities [van Amerongen & van Nauta Lemke, 1978], while at the same time attempting to improve fuel consumption. Factors such as wind and currents, or even the large inertia and slow response time of a ship should be taken into account when developing such systems [Yongqiang & Hearn, 2008]. With the development of Autonomous Underwater Vehicles (AUVs), autopilots designed for water vehicles need to operate in a three-dimensional world [Sutton & Craven, 1998] [Sarton, 2003]. Recent developments on autonomous sailboats bring forward new challenges for autonomous navigation at the surface [Cruz & Alves, 2008b] [Stelzer & Pröll, 2008].

Autopilot systems used by ground vehicles differ from application to application, according to the environment they are to operate on – a known environment (usually indoors) or an unknown environment (typically outdoors) –, the tasks it must perform (predefined path following or autonomous path discovery), whether or not it must interact with other vehicles, and many other aspects [Frazzoli et al., 2000] [Alm, 2002]. One of the most visible efforts in the development of autonomous ground vehicles is the DARPA Grand Challenge – a competition promoted by the US Defense Advanced Research Projects Agency (DARPA) for the technological development of autonomous ground vehicles. The two initial competitions, in 2004 and 2005, consisted of over 200km of desert roads and tracks, including tunnels and sharp turns; although no vehicle traveled

more than 12*km* in the 2004 edition, in 2005 five vehicles completed the course, with the winning vehicle doing it in under seven hours [Thrun et al., 2006]. In 2007, the challenge moved to an urban environment, and the vehicles had a 96*km* urban course where all traffic regulations should be respected; the winning vehicle completed the course in little over four hours [Urmson et al., 2007].

The idea of aircraft autopilot systems and their development started shortly after the first planes flew [Scheck, 2004]. Considering that a) the first sustained, controlled, powered heavier-than-air manned flight occurred in December 1903 and the first autopilot systems appeared only about a decade later; and b) boats and cars existed long before aircraft and autopilot systems for these vehicles were only later developed, it is easy to understand why aircraft autopilots are more advanced than autopilot systems for other vehicle types. In fact, several commercial aircraft feature autopilot systems which can control the aircraft from takeoff to land without human intervention (these systems are, however, usually only used during the leveled part of the flight, and during landing, when visibility conditions are below certain limits). These full autopilot systems use redundant computers to ensure that control decisions are correct, and provide a more stable flight than human pilots would, lowering fuel consumption at the same time.

With the development of Unmanned Aerial Vehicles (UAVs), some of which small in size, autopilot systems tend to also decrease in size, which means that additional challenges need to be faced.

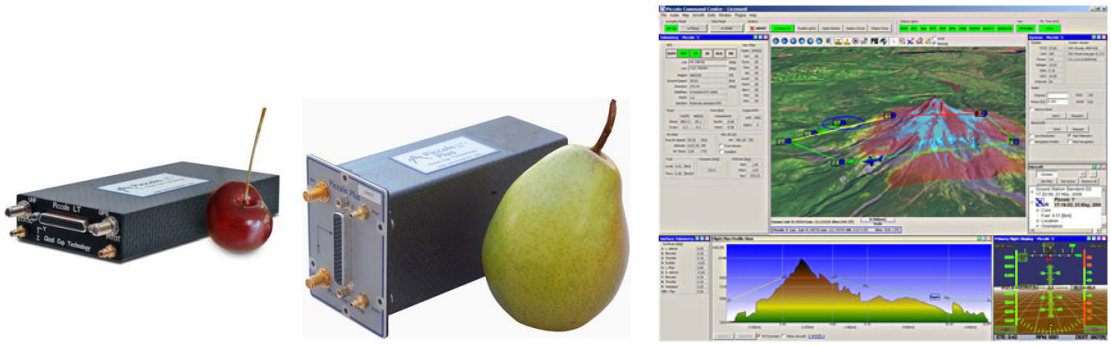
There is a variety of autopilots commercially available for small unmanned aircraft, usually comprised of light-weight hardware to connect to the aircraft, and a control station, also often including some software to interact with the system. These autopilots range from simple one-axis controllers to full three-axis systems, including redundant processors, sensor diagnostics and failure tolerance, medium- to long-range communication system and many other useful features when dealing with payloads [Chao et al., 2007]. Integration and testing of autopilot systems with the aircraft has to be carefully executed, as to calibrate the system to the aircraft in question [Erdos & Watkins, 2008].

Autopilot systems have also been developed for helicopters [Hoffmann et al., 1999]; some specific maneuvers and operations, such as formation flight [Lancaster, 2004] and in-flight refueling [Dogan et al., 2005] have also been studied by the community.

One example of an UAV autopilot system is provided by MicroPilot[26], as miniature UAV autopilots, weighing as little as 28 grams. Another example is Piccolo, aimed at small aircraft, and commercialized as a complete, off the shelf avionics system solution, including the core autopilot, flight sensors, navigation, wireless communication, and payload interfaces, all in a small, highly integrated package [CloudCap Technology, 2008]. It includes a software package (Piccolo Simulator) for software-in-the-loop development and tests as well as hardware-in-the-loop operations [Jager, 2008]. Figure 2.8(a) shows some Piccolo hardware components, and Fig. 2.8(b) shows the Piccolo Command Center, the main software interface to the system.

With the development and improvement of autopilot systems, along with the evolution in sensing and reasoning capabilities, several new vehicles have also been developed to operate in

[26]More information available online at `http://www.micropilot.com/`



(a) Piccolo Autopilot Hardware (b) Piccolo Command Center

Figure 2.8: Piccolo Hardware The Piccolo Command Center

situations where manned vehicles are not desirable (due to cost or danger involved) or even possible (unreachable places). These vehicles have an increasing level of autonomy, from earlier remotely operated vehicles, to vehicles that operate autonomously, but with pre-programmed paths and tasks, to fully autonomous vehicles. Several vehicles with varying levels of autonomy have been developed by or for the military, as to aid in missions where the presence of humans is not desirable [Sholes, 2007].

Three major types of autonomous vehicles can be identified, according to the means they operate on:

- **AUV (Autonomous Underwater Vehicle)**. AUVs are vehicles capable of autonomous navigation under water. Having evolved from remotely operated underwater vehicles, these vehicles vary in their degree of autonomy, but most of them are only capable of following a predetermined path [Dias et al., 2006] [Cao & Sun, 2008].

- **UGV (Unmanned Ground Vehicle)**. UGVs are vehicles capable of autonomous navigation on ground. Some UGV can only navigate on a known environment, while others are capable of navigating outdoors, in an unknown environment.

- **UAV (Unmanned Aerial Vehicle)**[27]. UAVs are vehicles capable of autonomous flight. A wide range of UAVs exist, depending on its intended capabilities and specific purposes for which it has been built.

Autonomous vehicles are designed for a number of different purposes, as can be seen in section 1.1. According to their main operational goal, autonomous vehicles can vary greatly in weight and size, speed, maximum operational altitude, range, endurance, speed, and several other aspects. As an example, the GlobalHawk UAV has a wingspan of almost $40m$ and is capable of flying at $65.000ft$ (almost $20Km$) with a range of over $20.000Km$ and an autonomy of approximately 36 hours. Also, the newer versions of the Predator drone have increasingly larger wingspans (both

[27]The nomenclature of the three major types can vary depending on author or areas of application. For instance, it has been proposed to rename UAV to UAS (Unmanned Aircraft System), or even UAVS (Unmanned-Aircraft Vehicle System), to reflect the fact that the system is comprised of more than simply the vehicle.

the Reaper and the Avenger have wingspans of $20m$, while Predator had less than $17m$), higher operational altitudes (the Avenger has an operational altitude of $60.000ft$, while the Reaper has an operational altitude of $25.000ft$, with a ceiling of $50.000ft$, and the Predator has a service ceiling of $25.000ft$), ranges (the Reaper has a range of almost $6.000Km$, when compared to the $3.700Km$ of Predator), endurance (Reaper has an autonomy of up to 28 hours, while Predator has an autonomy of 24 hours) and speed (Predator has a cruise speed of up to $165km/h$, while Reaper has a cruise speed that can go over $300km/h$ and Avenger over $740km/h$). On the other hand, the Wasp III UAV has a wingspan of less than $75cm$, a cruise speed of up to $65Km/h$, and a range of approximately $5Km$.

One of the most complex tasks to achieve when considering aircraft is formation flying. Professional pilots train very hard to achieve this in real life (for instance, the Blue Angels[28], one of the world's most recognized formation flying teams, or the Portuguese *Asas de Portugal*[29] receive intense training before they are able to fly in formation). When considering autonomous vehicles, this also consists in a challenge, that has been of interest for researchers for several years [Wang, 1989], [Desai et al., 1998], [Lancaster, 2004].

A controlled formation flying pattern is also necessary when performing certain operations, such as in-flight refueling. During this delicate operation, both the refueling tanker aircraft and the receiving aircraft must maintain a constant distance from one another, and move at the same speed [Dogan et al., 2005].

Generic swarming is somewhat easier to achieve, with the application of three simple rules [Reynolds, 1987]:

1. **Collision Avoidance**. Avoid collisions with nearby flockmates;

2. **Velocity Matching**. Attempt to match velocity with nearby flockmates;

3. **Flock Centering**. Attempt to stay close to nearby flockmates.

By adding some additional rules, such as obstacle avoidance or a desired flying direction, some more complex behaviors can be achieved [Kanchanavally, 2006].

A somewhat common approach to formation control is based on the notion of leader. This technique is based on one vehicle (the leader) having a planned motion, while the other vehicles are programmed to follow the leader, either directly or indirectly [Desai et al., 1998].

This, however, does not yet guarantee that the formation will maintain a desired geometry. For that, desired distances between members have to be taken into account, and each member of the team (not necessarily corresponding to a specific vehicle) is assigned a position within the formation. Several studies have been conducted in this area, with proposed solutions for achieving a stable motion while maintaining the desired geometric shape [MacArthur, 2006], [Marshall, 2005].

[28] More information available online from `http://www.blueangels.navy.mil/`

[29] More information available online at `http://www.asasdeportugal.com.pt/`

2.3 Specification Languages

The literature that can be found is diverse, when considering all aspects captured by the developed languages; however, few are capable of fully expressing the high-level concepts that these languages were designed to express (as presented below).

A standardized textual description of the layout of a given physical area in a format that can be easily read by a person is not always easy to find. Several applications exist, however, that show that same information in a graphical and intuitive manner.

One of the most notable languages used to describe geographical data is GML (Geography Markup Language), a standard developed by the Open Geospatial Consortium (OGC[30]) to express diverse geographical features [OGC, 2007a]. After several years of developments and changes, it has been approved as an international standard in 2007 and is being used by several applications[31] [Huang et al., 2009].

CityGML[32] is one of the most well-known applications of GML [OGC, 2008b]. It is a markup language used to describe three-dimensional urban objects and scenes; it was adopted as an OGC standard in August of 2008, and is used in several applications [Fan et al., 2009].

The Aeronautical Information Exchange Model (AIXM[33]) is an international data standard and a model that supports aeronautical information collection, dissemination and transformation throughout the data chain in a digital format [Brunk & Porosnicu, 2004]. It started as an initiative from Eurocontrol (the European Organization for the Safety of Air Navigation) over twelve years ago and had, later on, the active participation of other entities, such as the FAA (the US Federal Aviation Administration), ICAO (International Civil Aviation Organization), or NATO (North Atlantic Treaty Organization). It is currently in it's fifth version, adopting GML as a basis, and providing numerous features for several entities involved in the aeronautical industry [Brunner et al., 2007].

One of the foremost applications that uses a description similar to part of the one described below is flight simulators. Airport descriptions in flight simulators can be very detailed, allowing for a very realistic visual simulation. However, and for that reason, airport description tends to be mostly centered around visual aspects, such as signs, lights, lines, and many other aspects that, even though being very important for a visual simulation, do not contribute much for the goals of this platform and developed languages.

One of the flight simulators analyzed to more detail was Microsoft's Flight Simulator X. Airport information is contained in pre-compiled files, and features numerous visual aspects along with airport structure. Extensive documentation on the format and information contained within the files is provided, and an application to compile these files is included in the SDK (along with a Schema for the XML definition of the format) [Microsoft Corporation, 2008a]. Some tools have been developed by the community and software vendors to help interact with this simulator. One

[30] More information available at `http://www.opengeospatial.org/`
[31] More information available at `http://www.ogcnetwork.net/gml`
[32] More information about CityGML at `http://www.citygml.org/`
[33] More information available at `http://www.aixm.aero/`

such example is Airport Facilitator X (AFX), a product that provides a visual interface to design and edit airports and their elements, such as runways, taxiways, towers, parking spaces, and so on [Flight One Software, Inc., 2009]. Another example is the Airport Design Editor (ADE), which provides similar functionalities [Masterson et al., 2009].

Other largely known simulators have different representations of such information. For instance, FlightGear uses a format that allows for a compact representation of airport runways and taxiways, by using a series of letter-based codes to represent enumerations of surface types, signaling and lighting [Peel, 2001]. However, some information could be better represented (for instance, taxiways are not described as a whole, but by individual segments). Another flight simulator, X-Plane, uses a very similar representation, for airport structures [Peel, 2009]. The file specification is based on a series of codes (mostly numeric) and is more complete than the one from FlightGear. However, both representations are far from being easily read by a human without the aid of an interpreting application.

Most works found in the literature that deal with the definition of a team of vehicles focus on hierarchical or behavioral aspects, which are not here represented at the team level, but at the mission level. Furthermore, most of these works do not express the vehicles that compose the team or their capabilities in an explicit form, but rather include that information in an ad-hoc manner, in custom-developed formats, or even embedded within the application itself, thus reinforcing the need for the development of standard languages for team description.

As previously mentioned, one of the most visible applications is forest surveillance, in search for fires. Several research groups are currently working on fire detection using Unmanned Aerial Vehicles to carry several sensors; however, most of these project use real equipment, and not a simulator. An exception is the project described in [Casbeer et al., 2006], which uses the simulator described in [Hargrove et al., 2000]. This simulator considers several factors, such as fuel type and moisture, wind speed and direction, to determine how a fire spreads across a landscape previously divided in a grid, each cell 50m in side. However, no formal definition or description language could be found that can describe, using high-level concepts, fire size and spread patterns.

Another application is the discovery of underwater hydrothermal vents. Some work has been developed to automate the search for hydrothermal vents, using AUVs [Yoerger et al., 2002] [Jakuba, 2007]. However, and similar to what happens in the case of fire, most projects deal with real vehicles. Several publications report on the developments or use of vent simulators, such as [Saigol et al., 2010] (which uses a different approach from [Jakuba, 2007] for searching for hydrothermal vents), [Coumou et al., 2006], [Thomson et al., 2005] or [Subbotina et al., 2008], among others, even though they don't present any formal mechanism to model the vent, and its evolution patterns.

A similar problem is the description of the source of a polluting chemical and how it spreads, even though work exists regarding the detection of pollution (see section 1.3).

Applications such as search for fire, pollution source, or hydrothermal vent location require some sort of dispersion modeling, in order to simulate the dispersion of gases in the atmosphere (or chemicals in the ocean) [Turner, 1994], [Barratt, 2001]. This has been a research topic for many

years, and there are many models used by governmental and environmental agencies throughout the world[34]. Thus, this framework can also be used in testing different dispersion models, as to determine the differences among them and the adequacy of the models to each specific situation.

All these works, however, focus on the execution of the mission itself, but fail to provide with the information on how the disturbances to be detected and their behavior are modeled (the exception to this is the work that has been done on dispersion models).

Regarding a mission specification for a team, there have been several approaches over the year.

CHARON is a language used to describe the workings of agents, communicating via shared variables, the behavior of each agent modeled by a hierarchical state machine [Alur et al., 2000]. It has been used in some practical case studies[35], including formation control [Hur et al., 2003].

MALLET (Multi-Agent Logic Language for Encoding Teamwork) is a BDI-based language, based on predicate logic that allows for the encoding of teamwork [Yin et al., 2000]. It provides a number of predefined predicates that can be used to express how a team is supposed to work in each domain. It defines team members and existing roles, and the association between members and roles. It also defines several stages for a mission, allowing for different goals to be defined for each stage, mapping the roles involved in achieving each goal in each stage. It defines hierarchical plans, which decompose into actions (each with a set of pre-conditions and post-conditions), which are associated with the roles that can perform such actions. These concepts are then transformed into a Petri Net model, which are also used to determine the interactions between agents. This has been used mainly in the training of teams comprised of both autonomous agents and humans [Miller et al., 2000]. The description of the semantics of MALLET can be found in [Fan et al., 2005] and [Fan et al., 2006], along with a brief description of its implementation using CAST (Collaborative Agents for Simulating Teamwork), a team-oriented agent architecture that supports teamwork using a shared mental model.

Another approach uses CDL (Configuration Description Language) to describe a recursive composition of agent systems [MacKenzie et al., 1997]. It can describe low-level behaviors that can be reused to assemble higher-level ones, which can be temporally sequenced as finite state machines. The specification is independent of robot architecture, with the exception of primitive behaviors. CDL is then used to generate robot-specific code, maximizing code reuse and minimizing machine dependencies. It is implemented in MissionLab, a set of graphical tools that facilitates the specification process [Endo et al., 2006].

CCL (Common Control Language) was designed to establish a standard interface for information exchange and task delegation among agents [Duarte & Werger, 2000], [Duarte et al., 2004]. It has evolved over the last years, and was used some years ago to specify simple tasks for a group of AUVs [Duarte et al., 2005]. Currently, two layers are specified: Layer 1 – CCL Vocabulary and Message Set Specification – and Layer 2 – CCL Support Library[36].

[34] More information can be found online at `http://atmosphericdispersion.wikia.com/wiki/Main_Page`

[35] See `http://rtg.cis.upenn.edu/mobies/charon/CHARONpapers.html` for more information

[36] More information can be found online at `http://ausi.org/research/behavior/`

Some other languages have been developed for a more specific application, as is the case of COACH UNILANG [Reis & Lau, 2001a], developed for the soccer domain. Based on a set of concepts (that include field regions, time periods, tactics, formations, situations or player types), three levels of abstraction are defined for coaching soccer teams.

2.4 Summary

This chapter introduced some of the key concepts regarding several aspects of the work reported herein.

The first section focused on agents and multi-agent systems. First, the concept of agent is presented, along with the description of the agent environment and agent architectures. Then, an overview on multi-agent systems and coordination is presented, followed by a review of agent communication platforms and agent-oriented software engineering methodologies.

In the second section, some concepts regarding simulation were introduced, with an emphasis on flight simulation. An introduction to auto-pilot systems was also presented, describing their multiple applications in several vehicle types.

Finally, in the third section, some approaches to formally represent most of the elements and concepts described by the developed languages (see chapter 4) were presented.

Having provided with an overview of these three areas, the following chapter will introduce the architecture of the developed platform.

Chapter 3

Platform Architecture

This chapter introduces the general architecture of the proposed platform. First, a generic model for multi-robot systems is presented using the Gaia methodology (which was introduced in section 2.1.7). Then, this generic model is instantiated considering the problem at hand, and the architecture of the developed platform is described, followed by a description of the data flow model. Finally, an introduction is made to the four languages developed for this platform.

3.1 Generic Model for Multi-Robot Systems

Realizing that the developed platform has several points in common with the architectures of similar projects involving mobile robotic vehicles, a generic model has been developed that can be used as a basis for the specification of such systems [Silva et al., 2010], [Silva et al., 2011b].

After considering several possible agent-oriented software engineering methodologies, as presented in section 2.1.7, and also considering the research group's past experience with one of the methodologies [Castro & Oliveira, 2008], Gaia was chosen for expressing the model.

For a better understanding of the Gaia methodology, and more specifically of its second version (see section 2.1.7), a SPEM (Software Process Engineering Meta-Model) model for Gaia was produced, which can be found in Appendix A.

The Gaia methodology is here divided into four stages – Requirements Gathering, Analysis, Architectural Design and Detailed Design. Even though the first stage is not actually part of the Gaia methodology, it is present to introduce the general requirements common to most multi-robot systems, which are the basis for the remaining stages.

3.1.1 Requirements Gathering

There are several contexts in which a multi-agent system comprised of mobile robots can be of value (military [Future Combat Systems, 2008], medical [Katevas, 2001], industrial [Kroll, 2008], household [Krose et al., 2004] and many others [Silva et al., 2005]); most of these systems have

similar high-level requirements, which promotes the development of a common meta-model that aggregates all these requirements, and can be used as a basis for a more rapid development of specific system models.

The first and foremost requirement is the presence of mobile robotic agents. These vehicles typically have several sensors and actuators, as well as communication capabilities. They are usually autonomous, but most systems require the possibility to manually take over or share their control. These vehicles are usually used to perform tasks or missions, which may be delegated to a single robot or to a group of robots, to be performed in a distributed manner (in which case the vehicles should be able to cooperate, in order to optimize resources and improve mission performance). As with most multi-agent systems, communication between agents should follow the FIPA guidelines for agent communication[1], and a set of services should be made available, namely an Agent Management System, a Message Transport System and a Directory Facilitator [FIPA, 2004]. The vehicles operate in an environment, which is usually only partially accessible, dynamic and can be diverse in terms of structure, presence of other agents (either robotic or human) or level of intelligence of devices – for instance, intelligent doors, windows, lights, air conditioning, cargo load/unload system (robotic arm), among others. In addition to the robotic agents in the environment, several utilitarian agents are usually required, such as a logging mechanism, a centralized redundant system for detection and resolution of possible conflicts between two or more agents, and an interface with humans, through which tasks and missions can be specified.

As a result of the first stage, these requirements are collected and structured in a document that will act as the input for the following stages. Figure 3.1 shows the general architecture of the system, which may be seen as a summarized graphical representation of the requirements specification document.

3.1.2 Analysis

The analysis stage is comprised of five deliverables – the identification of the sub-organizations that constitute the system; the environment model; preliminary roles and interaction models; and organizational rules. The preliminary roles and interaction models however, being of preliminary nature, are not shown at this stage. Also, given the nature of these systems, and the intended generic nature of the model, organizational rules that can be applied to all intended domains are somewhat rare, and therefore should be determined separately for each system that derives from this generic model.

3.1.2.1 System Sub-Organizations

Given the nature of the systems to be implemented, the identification of sub-organizations is not possible (or even logical) from the Gaia standpoint, since there is no established hierarchy or organizational structure. On the other hand, the roles can be arranged into groups, with logical

[1] Current FIPA standard specifications can be found at `http://www.fipa.org/repository/standardspecs.html`

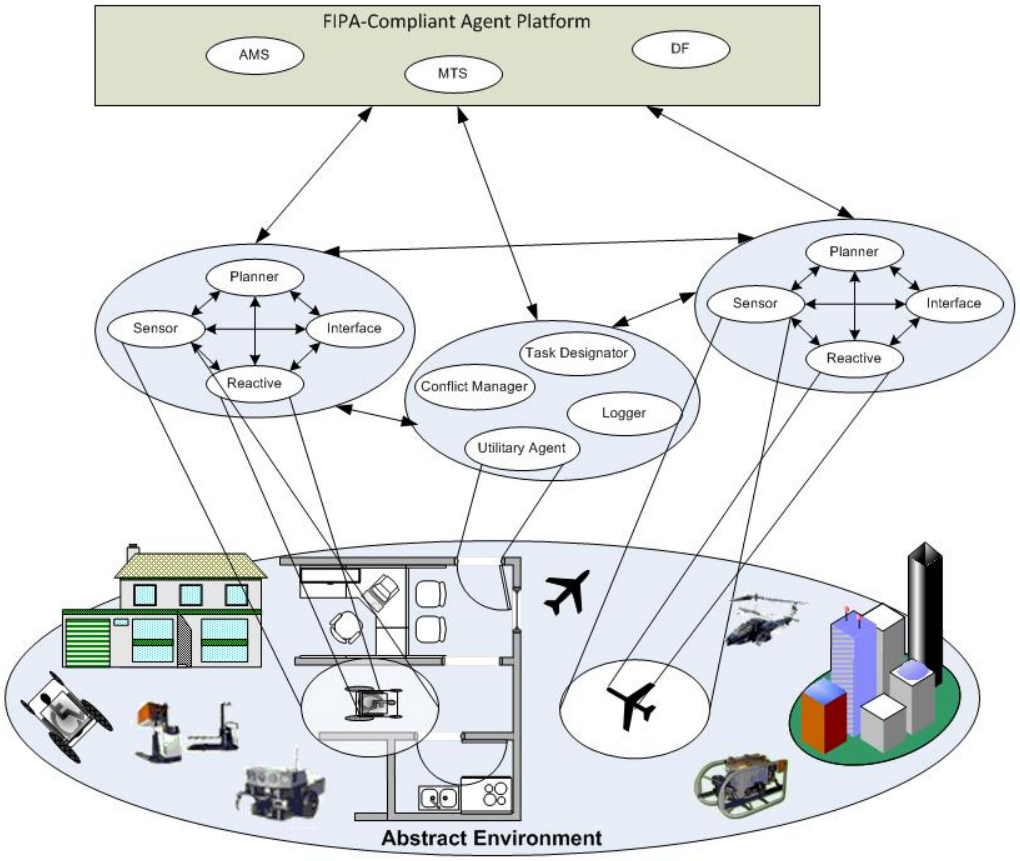


Figure 3.1: Generic Informal Platform Architecture

(or physical) similarities. In this system, two different groups of agents were identified. The first group, named Mobile Robot, includes four agents that compose and represent a mobile robotic platform. The second group, named Services, includes agents that perform several tasks within the system, at a global level. The agents in the first group are tightly-coupled (usually running on-board the robotic platform), while the agents in the second group are loosely-coupled. An instance of the first group interacts with other instances of the same group, and with the agents in the second group. These concepts are represented graphically in Fig. 3.2.

3.1.2.2 Environment Model

Since the environment in which these systems are intended to operate is the real world, with all its variables and uncertainties (and not a controlled, closed environment), a complete environment model is not suitable in this case. Figure 3.3 presents a general environment resources diagram, showing a preliminary version of existing roles and generic, common environment resources. Each system implementing this generic model should better describe the environment it will operate on, and the possible particularities of such environment.

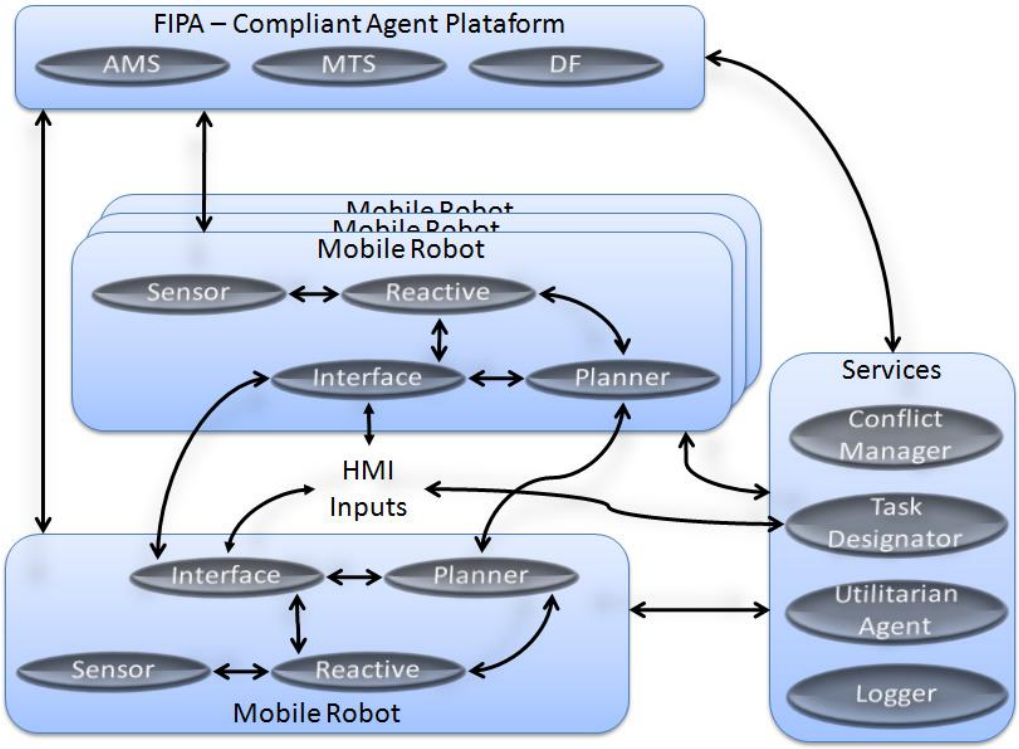


Figure 3.2: System Sub-Organizations as Groups

3.1.3 Architectural Design

The architectural design stage is comprised of three deliverables – organizational structure, the roles model and the protocol model – plus the roles and interaction diagram (as introduced by [Castro & Oliveira, 2008]). The organizational structure, similarly to the sub-organizations model, also does not apply to these systems, given that no fixed common hierarchical structure exists, and therefore, this model is not presented. Systems implementing this generic model that wish to include some degree of hierarchy or control structure between agents or between agents and the global services should include that information in the model.

3.1.3.1 Roles Model

A total of nine roles were identified – five of these roles are included in the Mobile Robot group and the remaining four have been clustered into the Services group.

The five roles that belong to the mobile agent platform are the Sensor, Reactive, Planner, Interface and Broadcaster roles.

The Sensor role is the most basic role, and is responsible for gathering all information about the environment, using sensor information, and updating the internal representation of the environment, so that other agents/roles can use it.

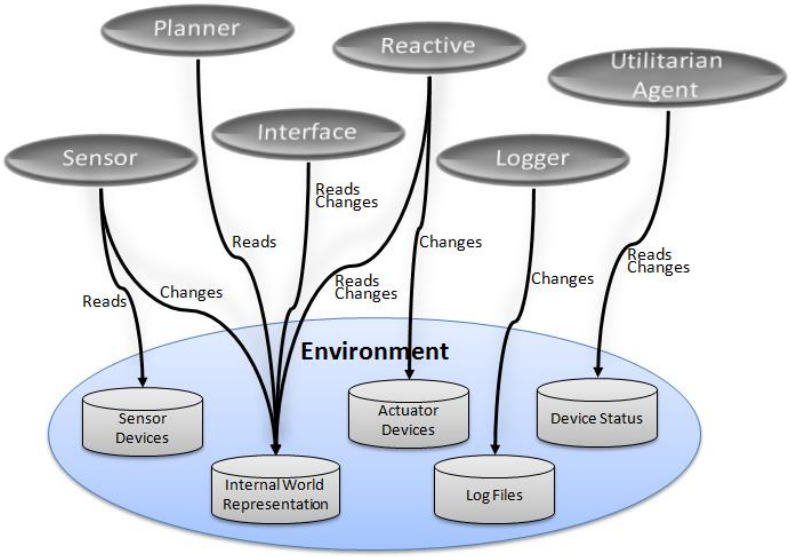


Figure 3.3: Environment Resources Diagram

The Reactive role (see Fig. 3.4(a)) is also a basic role, and is responsible for all low level control, using the internal representation of the environment together with low level goals for determining action control.

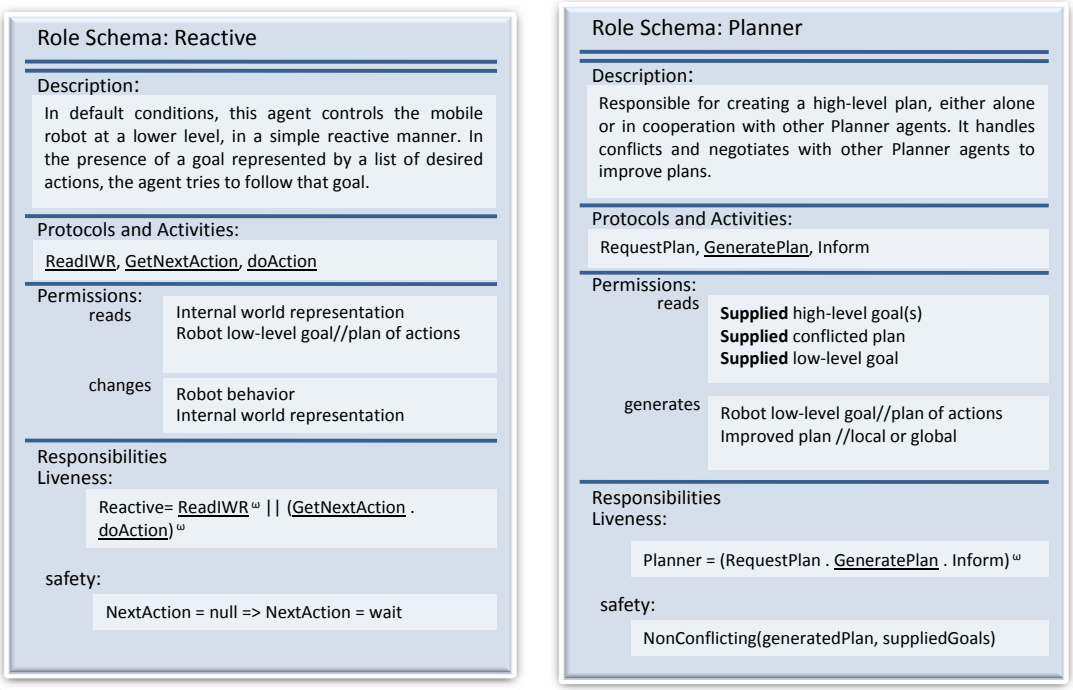
The Planner role is responsible for high level control, and is responsible for creating a sequence of high level actions needed to achieve the global goal. It also integrates a cooperation and collaboration facet between the robotic platform it represents and other robotic platforms – see Fig. 3.4(b).

The Interface role establishes the interface between user and robotic platform; this interface is where all the information is gathered, and where relevant information is displayed in real time. It also receives orders from users and forwards them to the appropriate agent. It should also allow the user to assume a manual control of the robotic platform it represents.

An agent implementing the Broadcaster role is responsible for broadcasting, at regular intervals, the internal world representation and state of the mobile platform it represents to the agents that subscribed to that information.Agents implementing roles such as Logger, Utilitarian Agent or Conflict Manager (detailed below) can subscribe to this information (by using the Broadcast protocol, presented below), and use it to update their knowledge about the several robotic platforms moving through the environment, and adjusting their actions accordingly.

The four roles outside the mobile robotic platform include the Logger, Utilitarian, Conflict Manager and Task Designator roles.The Logger role is responsible for creating a set of log files containing pertinent information regarding both the agents and the environment. The Utilitarian role may be instantiated in a number of agents, representing doors, windows, or other elements within the environment, so that these elements can interact with the robotic platforms, and in this way make the navigation through the environment easier.

The Conflict Manager is responsible for monitoring the environment and the mobile agents,



(a) Reactive Role (b) Planner Role

Figure 3.4: Reactive and Planner Roles

searching for possible conflicts or deadlocks – see Fig. 3.5(a). When one is found, the agent implementing this role is responsible for solving that conflict, either by enforcing a solution on the robotic platforms, or by cooperating with them in order to cooperatively find a suitable solution to solve the upcoming conflict.

The Task Designator role (see Fig. 3.5(b)) is responsible for providing human actors with a means to interact with the system as a whole. This allows them to specify the missions that should be carried out by the system (either by a single robotic platform, or by multiple platforms).

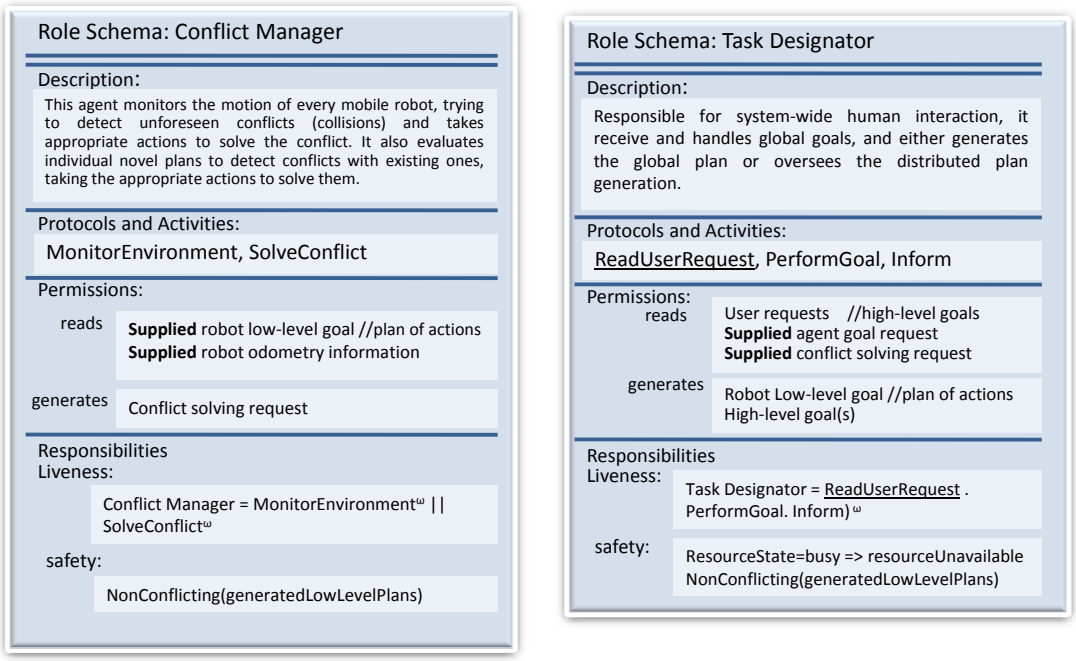
3.1.3.2 Protocol Model

A total of six protocols are presented in this meta-model, even though more protocols were identified in the complete version of the meta-model.

The Role Switch protocol can be used by any one of the Sensor, Reactive, Planner and Interface agents. This protocol is used to request a transfer of the Broadcaster role to another agent, and is usually triggered by an increase in the work load of the agent's core tasks – see Fig. 3.6(a).

The Broadcast protocol (see Fig. 3.6(b)) is used to broadcast information regarding the robotic platform and the environment so that agents subscribing to that information can receive the updated information.

The Monitor Environment protocol is used by agents outside the robotic platforms to subscribe to information about their state (given by the agent implementing the broadcaster role).



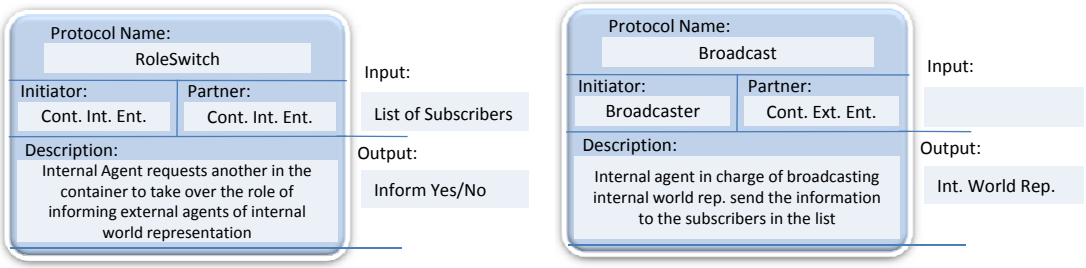
(a) Conflict Manager Role (b) Task Designator Role

Figure 3.5: Conflict Manager and Task Designator Roles

The Solve Conflict protocol is initiated by the agent implementing the Conflict Manager role, when a conflict is detected and the agents are supposed to cooperate in solving the conflict. This protocol is used to communicate with the Planner agent of each robotic platform involved in the conflict, and aims at solving it, by reaching a compromising solution.

The Request Plan protocol is initiated by the Interface agent and enables it to request the Planner agent to devise a plan that can lead the robotic platform to achieve the supplied high-level goal.

The Perform Goal protocol is usually initiated by the Task Designator agent and enables it to ask the Interface agent of a specific platform if the platform can generate a plan that can be used to achieve a given high-level goal.



(a) Role Switch Protocol (b) Broadcast Protocol

Figure 3.6: Role Switch and Broadcast Protocols

3.1.3.3 Roles and Interaction Diagram

For a better understanding of all roles and interaction protocols present in the system, a Roles and Interaction Diagram, as proposed by [Castro & Oliveira, 2008] is presented in Fig. 3.7. The roles identified in the system are presented as classes and the protocols between them are presented as associations, including the direction in which the protocol is activated.

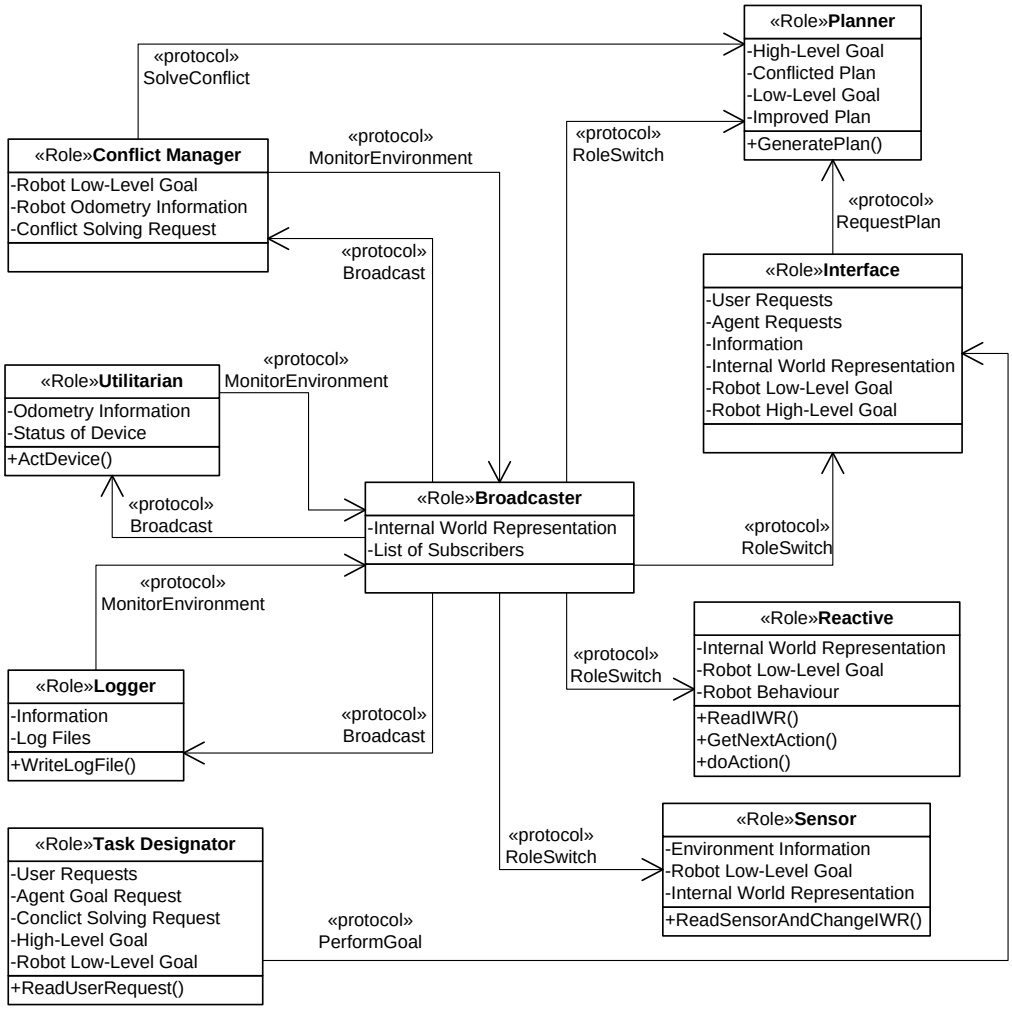


Figure 3.7: Role and Interaction Diagram

3.1.4 Detailed Design

The detailed design stage is comprised of two deliverables – the agent model and the services model.

3.1.4.1 Agent Model

The agent model can be seen as a mapping between agents and roles, indicating how many instances of each agent will exist in the system, and which roles each agent will implement – see

Table 3.1. In this particular case, the four agents that represent the robotic platform (Sensor, Reactive, Planner and Interface) will have N instances, corresponding to the number of robotic platforms in the system. The Utilitarian Agent can have up to U instances and the Conflict Manager can have up to C instances. One should also point out that even though the Broadcaster role is present and implemented by the four agents internal, only one of these agents will implement the role at any given time.

Sensor $^{1..N}\overset{play}{\rightarrow}$ Sensor, Broadcaster
Reactive $^{1..N}\overset{play}{\rightarrow}$ Reactive, Broadcaster
Planner $^{0..N}\overset{play}{\rightarrow}$ Planner, Broadcaster
Interface $^{0..N}\overset{play}{\rightarrow}$ Interface, Broadcaster
Utilitarian Agent $^{0..U}\overset{play}{\rightarrow}$ Utilitarian Agent
Conflict Manager $^{0..C}\overset{play}{\rightarrow}$ Conflict Manager
Task Designator $^{0..1}\overset{play}{\rightarrow}$ Task Designator
Logger $^{0..1}\overset{play}{\rightarrow}$ Logger

Table 3.1: Agent Model

3.1.4.2 Service Model

The service model is intended to identify the services associated with each agent class or role. As proposed by [Castro & Oliveira, 2008], the service model table was replaced by a UML class diagram – the missing information (output) was also included as notes to the services in the diagram. Figure 3.8 shows a few services provided by the system.

3.1.5 Summary

This generic model provides a common base work, that can be used as a basis for the specification of several systems. One system that makes use of this generic model can be found in [Braga, 2010]. The system described in this thesis is also one of those systems.

The Agent in the platform architecture (Fig. 3.9) gathers the five roles defined in the meta-model for each autonomous robot – Sensor, Reactive, Planner, Interface and Broadcaster. The Task Designator role is mapped onto the Control Panel, responsible for interacting with the user in the definition of tasks and missions. The Conflict Manager role is mapped onto the ATC agents, as a spatially distributed task (each instance is responsible for a portion of space). The Logger role has the obvious mapping onto the Logging tool. Finally, the Utilitarian Agent role is mapped onto the Disturbances Manager.

In respect to the models produced by this methodology, some adaptations had to be made in order to better fit the systems to be modeled. In more detail:

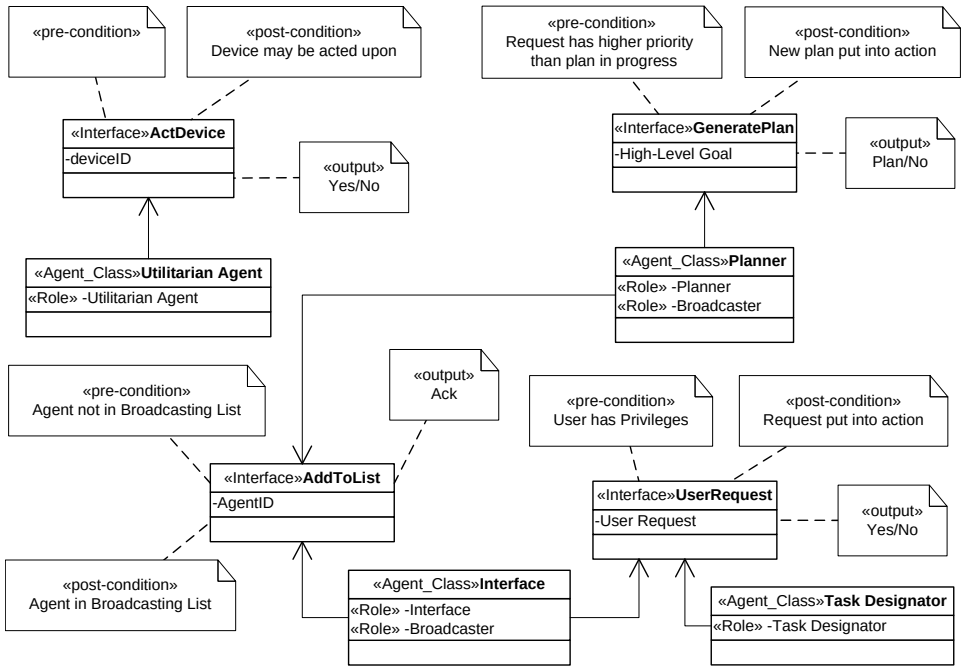


Figure 3.8: Service Model

- Regarding the System Sub-Organizations, since most of the systems intended for modeling do not possess an hierarchical or otherwise structured organization, this model was adapted as to reflect how the different roles may be grouped (logically or physically), possibly in different platforms (mobile or otherwise).

- Regarding the Environment model, since these systems operate on the real world, a model representing the environment would not be suited, and therefore should be included in each particular system implementing this meta-model if particular observations are required.

- The Organizational Rules that can be identified as corresponding to all systems being modeled are very few, and therefore each particularization should provide with an Organizational Rules model that includes the corresponding rules.

- The Organizational Structure model is also not suited for a meta-model, since different systems may have different hierarchical and control structures, or none at all, and therefore each system should provide its own model.

As for the Gaia process, the adaptations to designing open systems such as the ones depicted herein should also be included in a formal model that can be reused. These changes in the adopted version of the Gaia methodology could be included as a variation point in the methodology, according to the type of system being modeled [Webber & Gomaa, 2004]. Concerning the meta-model itself, it could also be further detailed, and the inclusion of variability points is also being discussed. These variability points would increase the model's flexibility and would allow it to be used with a wider range of systems.

Gaia's higher level of abstraction, when compared to other methodologies (other methodologies, and as presented in section 2.1.7, include the definition of implementation details in the final stages, while Gaia does not), proved to be an asset when designing the meta-model, and the adaptations that provide support for the design of open systems (as opposed to organizational-based systems) is believed to be a good contribution. Based on the authors' experience (both on modeling distributed systems and the ones described herein, using the meta-model as a basis) and the feedback from the research laboratory they are inserted in, using the meta-model as a basis has proved to be very helpful in the design of the distinct systems, by significantly reducing most of the common design tasks, which also reduces implementation difficulties, while at the same time providing a high-level overview of the system as a whole.

3.2 General Architecture

Based on the model presented above and the considered specific requirements, a global architecture for the proposed platform was devised – this architecture is represented in Fig. 3.9.

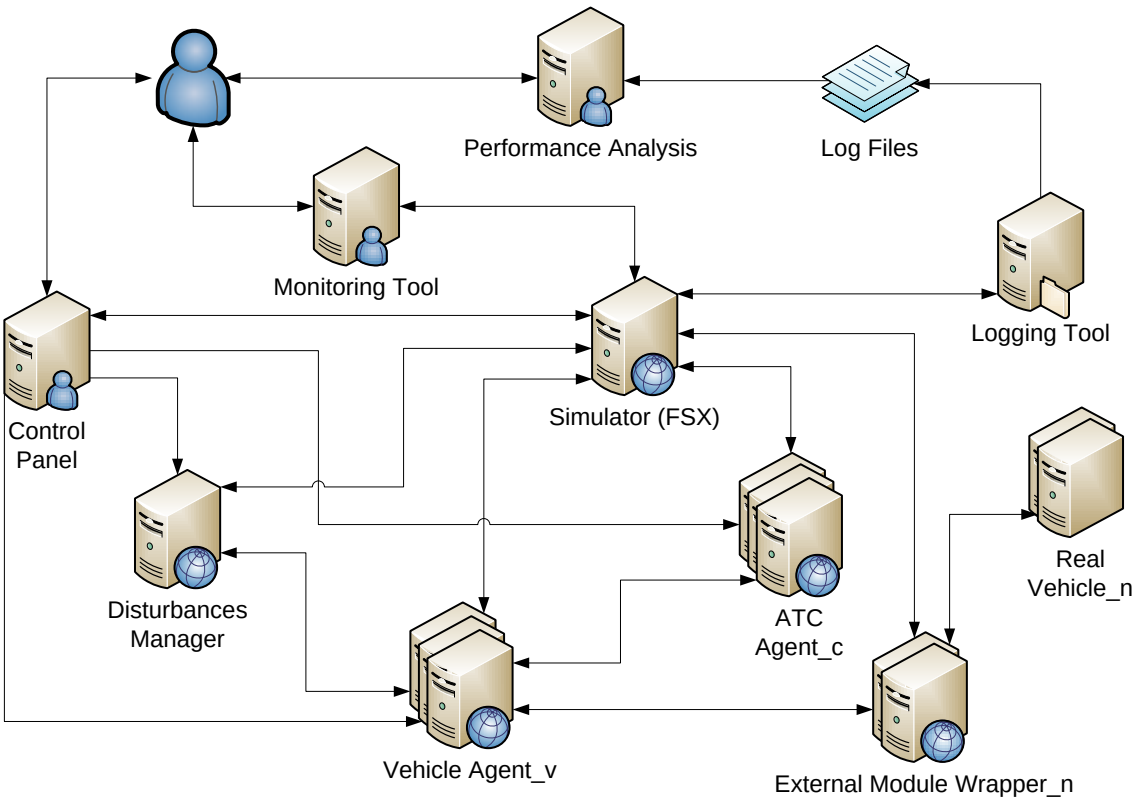


Figure 3.9: General Platform Architecture

The visual simulation platform acts as a central module for the system, given that it interacts with most of the other modules. The simulation platform should be able to provide with a realistic environment simulation and also be able to simulate several vehicles. The choice of the simulator,

as well as a description of its characteristics, and an explanation of some adaptations that had to be made is presented in more detail in section 5.1.

The Control Panel has a central role in the system – it interacts with the user and is responsible for system configuration, environment, disturbances, teams and missions definition and loading, and also for providing with system-wide status monitoring during mission execution – see section 5.2 for a more detailed description of the Control Panel. Configurations for scenario, teams, disturbances and missions use files expressed in the developed languages, detailed in chapter 4. After successful configuration of each of these components, other agents (namely ATC agents and Vehicle Control Agents) are created, and information is sent to the appropriate agents – see section 3.2.2 for more details.

The ATC Agent (described in more detail in section 6.2) is responsible for ground, air or sea operations in the vicinities of a base of operations. This agent represents the typical air traffic controller present in airport towers, responsible for a central control of all traffic on and around the airport, routing all ground traffic from or to the defined landing or departure runway, and avoiding traffic conflicts. Several instances of this agent may exist, each controlling a specific area and/or type of traffic (land/water/air). It is created by the Control Panel (see section 3.2.2), according to controller configurations – refer to section 4.2.6 for a detailed description of the configuration of a controller.

The modules identified as Vehicle Agent*1* through Vehicle Agent*V* represent the several (simulated) vehicles that exist within the system. Each one of these agents represents a vehicle, and is responsible for handling actions such as navigation control, collision avoidance, and others (more information can be found in section 6.3). It is created by the Control Panel (see section 3.2.2), according to configuration of vehicle type and specific vehicle – refer to sections 4.2.7 and 4.3.1 for a description of vehicle type and specific vehicle characteristics.

Given that the application is intended to be used with both simulated and real vehicles, there is the possibility to use external modules, which communicate with the robotic agents represented by the simulated vehicles. These modules act as wrappers between application actions or commands and specific vehicle functionalities. They will also allow for the collection of real-world vehicle data that will both replace the simulated data, if discrepancies are detected, and serve as the input to a calibration process that improves simulation realism. One of these modules exists for each simulated vehicle that also has a real counterpart.

The Disturbances Manager is responsible for creating and maintaining all disturbances within the simulator in accordance to their specification, and also for providing the interface between vehicle agents and disturbances, when the simulator cannot do so – see section 5.3.

The Monitoring Tool is responsible for providing both a real-time visualization of the status of the simulation and the agents, and also the updated values of several simulation and agent-related variables – see section 5.4.

The Logging Tool is responsible for creating permanent log files for each simulation session, including general simulation configurations and parameters, the initial simulation status, communications among the several agents, and a detailed status description for each agent (see section

5.5). These log files can then be used by the Performance Analysis Tool to provide the user with aggregated information regarding the simulation, and some analysis on the performance of a given team in completing a mission (more details in section 5.6).

In a metaphorical comparison, the Control Panel can be seen as an airliner operations center, the vehicle agents as representing aircraft pilots, the ATC agents as the air traffic controllers, the logging tool as the aircraft's flight data recorder (more commonly known as the black box), and the monitoring tool as a real-time flight tracker.

Each of the components of the platform is detailed in chapters 5 and 6.

3.2.1 Model Equivalency

This architecture is an instantiation of the generic model presented in section 3.1, even though a direct naming equivalence is not used.

The Vehicle Agent in the platform architecture (Fig. 3.9) gathers the five roles defined in the generic model for each autonomous robot – Sensor, Reactive, Planner, Interface and Broadcaster. The Task Designator role is mapped onto the Control Panel, responsible for interacting with the user in the definition of tasks and missions. The Conflict Manager role is mapped onto the ATC Agents, as a spatially distributed task (each instance is responsible for a portion of space). The Logger role has the obvious mapping onto the Logging tool. Finally, the Utilitarian Agent role is mapped onto the Disturbances Manager. In addition to the roles defined in the generic model, the Performance Analysis Tool was introduced as a means to provide the user with aggregated information regarding the simulation, and also as a possible means to calibrate the simulation platform and thus improve the overall performance of the team.

3.2.2 Data Flow

Figure 3.10 shows a simplified model of the data flow in the platform, with each step of the enclosed numeration described below. Unidirectional arrows represent either information being sent to a component (steps 1, 4, 7, 10 and 11) or components being created by the Control Panel (steps 2, 5 and 8), while bidirectional arrows represent communications between components (steps 3, 6 and 9). It is important to note that the Simulator (FSX) and the agent communications platform (AgentService) should already be running when the Control Panel is executed. Also, generic platform configuration should be provided in the Control Panel (these configuration details are saved in the registry from one simulation to the next, to expedite configuration procedures). At that time, the Control Panel connects to FSX and AgentService, to ensure that they are running in the specified network locations.

1. The scenario is configured. This is done in the Control Panel, using a previously created scenario file, or by creating a new one;
2. When the scenario configuration is launched, vehicle types are matched to simulator vehicles. New folders are created for each new vehicle type (when not already present in the

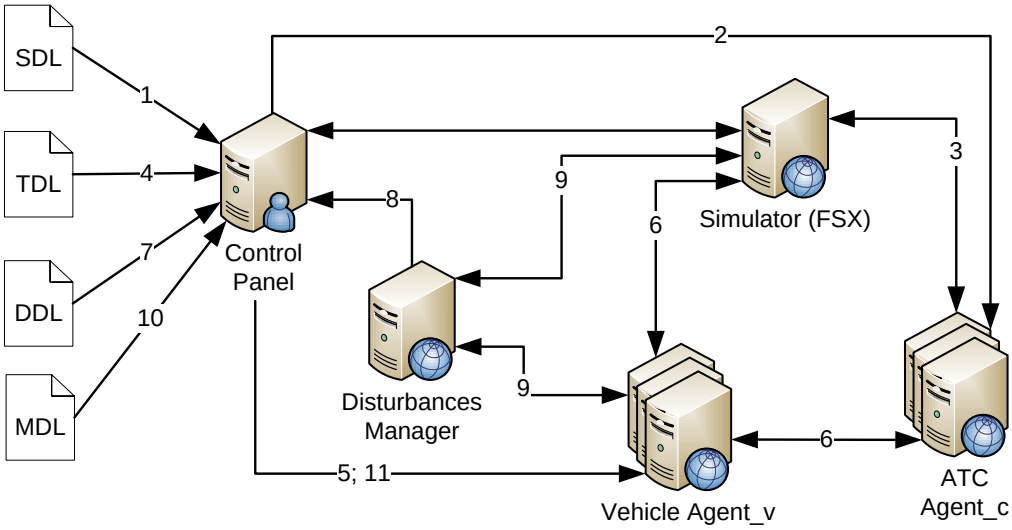


Figure 3.10: Simplified Data Flow Model

simulator), and vehicle details are modified in the respective configuration files. Also, ATC agents are launched according to controller configurations;

3. Each ATC agent connects itself to both the agent communications platform and the simulator, searching for vehicles within its defined area;
4. The team is configured. This is done in the Control Panel, also using a previously created team file, or creating a new one;
5. When the team is launched, all agents representing vehicles are created according to team and vehicle configurations;
6. Each Vehicle Control Agent loads the respective simulated vehicle into the simulator. Aware of their location, vehicles may establish a connection with an ATC agent, if within a controller area;
7. The disturbances are configured. This is done in the Control Panel, again using a previously created disturbances file or creating a new one;
8. When the disturbances are launched, the Disturbances Manager is created, and the disturbances description file is sent to it;
9. The Disturbances Manager loads all disturbances and in turn creates the appropriate objects within the simulator; it also communicates with the Vehicle Control Agents, if they are in sensing range of any of the disturbances;
10. The mission is configured in the Control Panel, using a previously created mission file or creating a new one;
11. Finally, when the mission is launched, the mission file is sent to all Vehicle Control Agents, which in turn start to communicate, coordinating actions to perform the mission.

3.2.3 The Description Languages

This section provides a brief description of the four languages developed for the configuration of a mission to be executed by a team of vehicles.

As mentioned before, four languages have been created. These languages were divided into two categories – static and dynamic. The static category encompasses the specification of scenario and teams, while the dynamic category encompasses the specification of disturbances to the environment and the missions to be performed. In addition to these two categories, the languages can also be classified according to their emphasis on either the scenario or the team. The scenario-oriented languages include both scenario and disturbances specifications, while the team-oriented languages include team and mission specifications. Table 3.2 shows the classification of the four languages according to category (static vs. dynamic) and emphasis (scenario vs. team).

	Static	**Dynamic**
Scenario	SDL	DDL
Team	TDL	MDL

Table 3.2: Language Classification

Each language focuses on different but related aspects:

- **Scenario Description Language (SDL)**. SDL specifies the static components of the environment, namely bases of operations (including airport, port and ground base), global constraints (namely no-fly areas) and control structures (such as traffic controllers), and a description of all existent vehicle types;

- **Teams Description Language (TDL)**. TDL provides with information about a team, describing the vehicles that compose the team and their specific details, as well as constraints that apply to the specific team (such as team-specific no-fly areas);

- **Disturbances Description Language (DDL)**. DDL specifies the dynamic components of the environment (such as a fire, vehicle, person, a source of pollution, and others), which usually require detection or other action(s) to be taken by the team;

- **Mission Description Language (MDL)**. MDL specifies a mission that should be performed by a team of vehicles, using high-level concepts, and allowing for soft and hard constraints to be specified.

The specification of the four languages uses high-level concepts and tries to abstract as much as possible from any operational details. This will help users with the specification of missions, but at the same time requires that the components of the platform be able to break down the high-level abstractions into an operational planning.

3.3 Summary

This chapter described the generic architecture for the developed platform. First, a generic model for platforms comprised of autonomous robotic agents was presented in section 3.1, using a modified version of the Gaia methodology, that accounts for non-hierarchical systems and the non-specific nature of such generic model. This generic model is, in its own, a contribution that enables system designers to save time, by providing a basis from where to start their work. It prevents repetitive tasks, and at the same time allows system designers to maintain a global perspective of the system.

Then, in section 3.2, the general architecture of the proposed platform was presented as an instantiation of the previously presented generic model. A brief description of each of the main components of the platform and their functions and interactions was provided, as well as the generic data flow model for the platform, which contributes to a better understanding of the dynamics of the platform. These components and their workings are described in more detail in chapters 5 and 6.

Finally, in section 3.2.3, an introduction to the four developed languages was made, focusing on their decomposition as scenario- vs. team-oriented and static vs. dynamic, and also briefly describing each of the languages.

The following chapter will describe the implementation details and the definition of each of the four languages.

Chapter 4

Description Languages

As mentioned in the previous chapter, the configuration of the platform is achieved mainly by means of four configuration files, that describe the configuration of scenario, teams, disturbances and missions. In this chapter, the formal languages developed to express the information regarding those topics are presented in detail. First, a brief description of some implementation considerations is made and then each of the four languages is described.

4.1 Implementation

The four languages were implemented as four XML (eXtensible Markup Language)[1] dialects [W3C, 2008], and specified using XML Schema [W3C, 2004].

A markup language, such as XML, allows for structure (hierarchical mainly) to be easily represented, and, being a self-documenting format, it provides not only with the data, but also with the meta-data to describe the data. Current technological advances (mostly regarding processing and memory capabilities) make the two major disadvantages of using such formats – the larger size of the resulting files and the processing costs associated with these documents – to be only minor disadvantages, which can, some of the times, even be overlooked. XML is perhaps the most popular markup language, and since its inception, it has been adapted to a wide range of applications, in a large number of areas, including medical [Schweiger et al., 2005], [Kumar et al., 2009], multimedia [Deursen et al., 2007], corporate [ANSI/AIIM, 2009] or military [Hobbs, 2003] [Wittman Jr., 2009], among other more traditional areas for XML applications, such as the web [Silva et al., 2007a], [Silva et al., 2007b], [Georgieva & Georgiev, 2010].

The specification of the dialects also needed to be used in the various components of the platform. For that purpose, the dialect specifications, in XML Schema, were converted into C♯ classes, using the XML Schema Definition Tool [Microsoft Corporation, 2010]. This tool is capable of generating Common Language Runtime classes based on an XML Schema document (it

[1] More information available online, from `http://www.w3.org/XML/`

can also generate an XML Schema file based on either a set of classes, or an XML file, making it a very flexible tool to be used when working with XML and XML Schema and one of the several supported programming languages). This easiness of transforming a dialect specification into code is also of great usefulness when changes are made to a dialect specification – the changes to the specification are rapidly reflected into the code, which reduces the cost of changes to the language specifications in the future. There is, however, one limitation to this process – no two elements can have the same name. Even though XML Schema allows for elements to have the same name but different specification (provided that an in-line type definition is used), this would imply the creation of classes with the same name, and therefore no two elements in the four dialects have the same name (except when using the same definition).

Part of the specification process for the four dialects was done using Altova XMLSpy[2], a powerful and flexible tool for working with XML-based technology. The Visual XML Schema Editor included in this tool was also used for capturing the images used to illustrate dialect specifications in this document.

4.1.1 Physical Structure

Each of the four dialects was specified in a separate XSD file. In order to facilitate the development of these dialects, elements common to two or more of them were grouped into a file (Common.xsd) that was then imported by each of the four main dialect specification files – see Fig. 4.1.

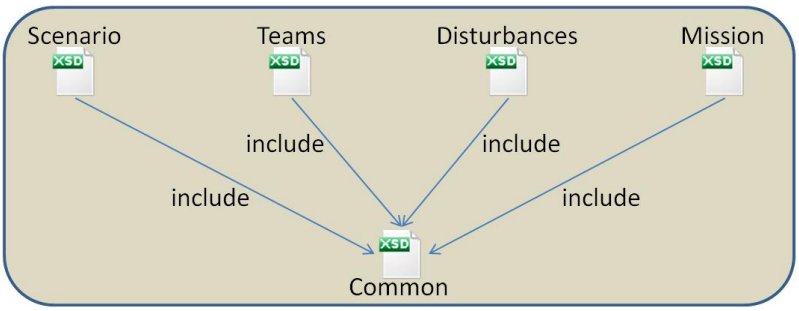


Figure 4.1: Physical Structure of XSD Files for Dialect Specification

This decision to eliminate repeatability was made as to facilitate specification and decrease the probability of introducing errors: using repeated code on more than one location increases the chances for an error to be made, especially if changes to an element need to be replicated into all definitions. This also allows for an element to be used in more than one dialect, without the need to change its name. This choice, however, also entails an increased need for attention during development – if a common element needs to be changed only in one of the dialects, but not in the other, it has to be removed from the common file and renamed. Table 4.1 shows the classification according to category and emphasis, as described above, but considering the five produced files.

[2]More information available online at `http://www.altova.com/xmlspy.html`

	Static	Dynamic
Scenario	SDL	DDL
	Common	
Team	TDL	MDL

Table 4.1: Language Files Classification

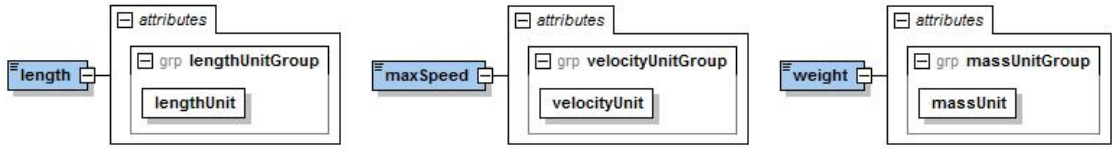
4.1.2 Additional Notes

All elements that provide a physical measure are accompanied by an attribute that defines the unit in which the value is being specified. This is done for two main reasons: first, to accommodate the different units that are commonly used to specify the same measurement type, according to context (ground vehicle speeds are usually indicated in either kilometers per hour or miles per hour, but for boats or aircraft the most widely used unit is knots); and second, to make the developed dialects compatible with users with different backgrounds (for instance, users in the UK and the US usually use different measurement units). Table 4.2 shows the units used in some measurement elements, as well as the symbols used for the representation of the unit in the Control Panel interfaces.

Type	Unit Name	Interface Symbol
Length	Meter	m
	Centimeter	cm
	Kilometer	km
	Foot	ft
	Inch	in
	Mile (Statute)	mi
	Nautical Mile	nm
Volume	Cubic Meter	m3
	Cubic Centimeter	cm3
	Cubic Kilometer	km3
	Cubic Foot	ft3
	Cubic Inch	in3
	Cubic Mile (Statute)	mi3
	Cubic Nautical Mile	nm3
	Gallon	gal
	Liter	l
Angle	Degree	Deg
	Radian	Rad
Area	Square Meter	m2
	Square Centimeter	cm2
	Square Kilometer	km2
	Square Foot	ft2
	Square Inch	in2
	Square Mile	mi2
	Square Nautical Mile	nm2
Mass	Gram	g
	Kilogram	kg
	Ounce	oz
	Pound	lb
Speed	Meters per Second	m/s
	Inches per Second	in/s
	Kilometers per Hour	kph
	Miles per Hour	mph
	Nautical Miles per Hour (Knots)	kts

Table 4.2: Measurement Units

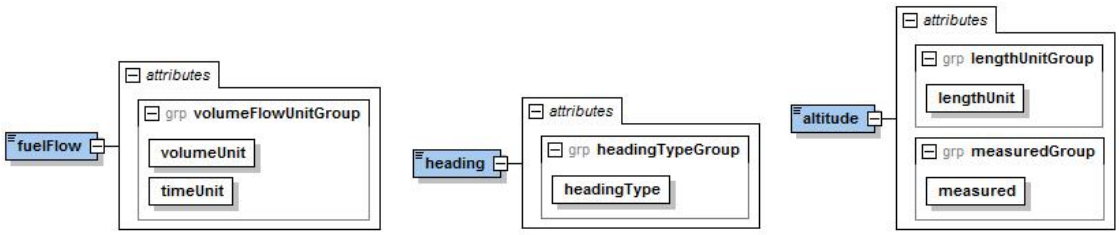
Most of these elements are located in the file that defines the common elements. Figures 4.2(a), 4.2(b) and 4.2(c) show examples of elements that use these attributes, namely the use of length, speed and mass unit attributes. Figure 4.3(a) shows an example of an element that needs two unit attributes, as it specifies the fuel consumption (volume over time).



(a) Length Element (b) Max Speed Element (c) Weight Element

Figure 4.2: Length, Maximum Speed and Weight Elements

Additionally, some elements feature some contextual choices, in a manner similar to unit choice; for instance, the altitude of a set of coordinates can be specified to be above the mean sea level (amsl) or above ground level (agl); a heading (direction) can be specified to be relative to the local Earth's magnetic field North orientation, also known as magnetic North (Mag) or relative to the Earth's geographic North, also known as true North (True). Figures 4.3(b) and 4.3(c) show two examples of elements that use these contextual choices – Fig. 4.3(c) shows an element (altitude) that combines the contextual choice attribute (how the altitude is measured) and the unit attribute (length unit).



(a) Fuel Flow Element (b) Heading Element (c) Altitude Element

Figure 4.3: Fuel Flow, Heading and Altitude Elements

4.2 Scenario Description Language

In this section, a full overview of the Scenario Description Language (SDL) is given, and each part of the scenario definition is analyzed in more detail. As previously stated, this dialect was developed in order to fully describe an operating scenario for a team (or several teams) of mobile robotic vehicles. The root element of a scenario description is named *scenario*, and it contains elements describing the unchangeable (static) part of the environment. Equation 4.1 formalizes the representation of a *scenario* element as a tuple constituted by four sets – bases of operations, controllers, agent types and no-fly areas.

$$
\begin{aligned}
Scenario &= \langle B, C, T, N \rangle \\
B &= \{BaseOfOperations\} \\
C &= \{Controller\} \\
T &= \{AgentType\} \\
N &= \{Area\}
\end{aligned} \tag{4.1}
$$

In more detail, the scenario is defined as a tuple comprised of:

- a set of bases of operations $B = \{b_1, b_2, \cdots, b_{nb}\}$ (which may contain an airport, port and/or a ground base) that can be used by one or more of the teams
- a set of controllers $C = \{c_1, c_2, \cdots, c_{nc}\}$, that control traffic on a well defined area of space
- a set of agent types $T = \{t_1, t_2, \cdots, t_{nt}\}$, that define the available vehicles types and their characteristics
- a set of no-fly areas $N = \{n_1, n_2, \cdots, n_{nn}\}$, that define the areas that no team can navigate through

Figure 4.4 shows the graphical representation of the *scenario* element.

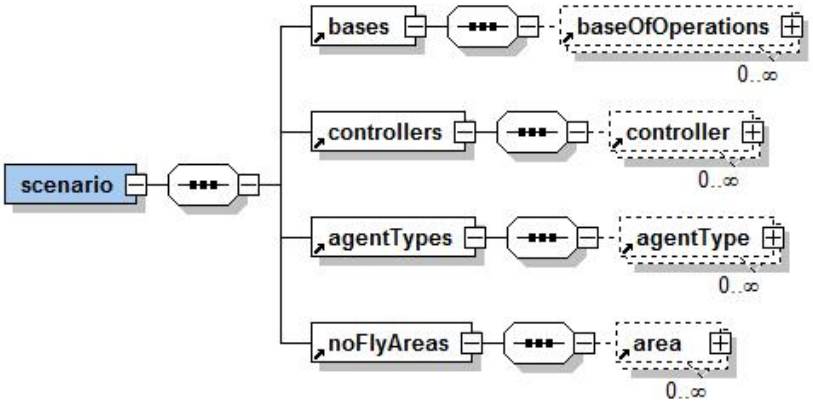


Figure 4.4: Root Scenario Elements

Each of the four main scenario elements is described in more detail in the following four sections – for presentation simplification, the *area* element (no-fly areas) is presented before the *controller* element.

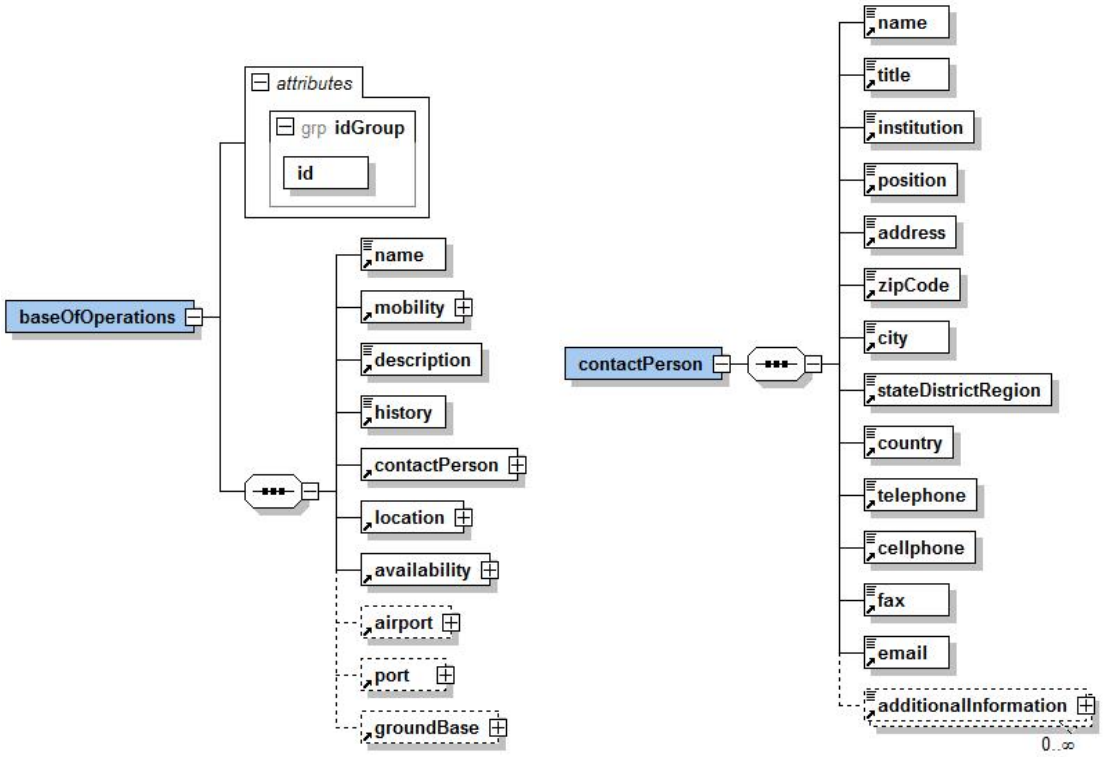
4.2.1 Bases of Operations

This section of the SDL file contains a list of available bases of operations. The concept of base of operations is derived from traditional military bases, which are well-defined regions (sometimes

even physically delimited) that contain structures and other resources that allow for services to be rendered to personnel, vehicles or other equipment.

Each base of operations has a unique identifier, which will later be referenced by the TDL file – see section 4.3. For each base of operations, a number of information details are provided (Fig. 4.5(a) shows the information contained within the *baseOfOperations* element in a graphical notation):

- **name**. The name by which the base of operations is known. If the base contains only an airport (or only a port, or only a ground base), the name of the base is usually coincident with its name.
- **mobility**. This element includes four boolean attributes (air, land, water and underwater), which indicate the type of vehicles the base provides support for – see Fig. 4.6(a). Also, these attributes define the existence of the optional elements *airport*, *port* and *groundBase* (described below).



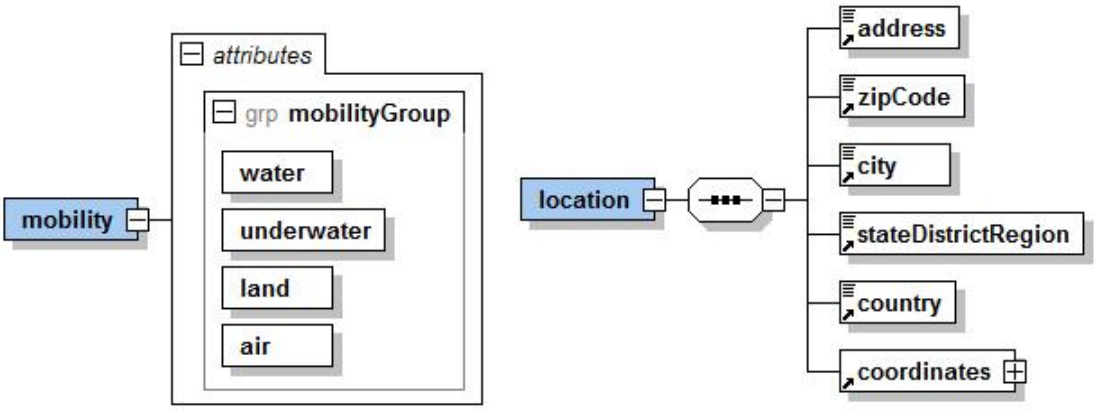
(a) *BaseOfOperations* Element Definition (b) Contact Person

Figure 4.5: Base of Operations and Contact Person Elements

- **description**. A brief textual description of the base of operations, that may include a listing of available services and facilities.
- **history**. This element can contain a detailed description of the base of operations, and its historical background, as well as the modifications it went through over time.

- **contactPerson**. This element contains detailed information about the person to contact regarding the base of operations. It includes name and title of the person of contact; the institution he works for and his position within that institution; address for physical correspondence (address, zip code, city, state and country) and other contacts (e-mail, telephone, cell phone and fax); and the possibility to add any additional information items, such as preferred contact hours or alternative contacts. Figure 4.5(b) shows the definition of this element.

- **location**. Provides information regarding the location of the base of operations. It is comprised by a physical address (which may or may not coincide with the address of the person of contact) and the coordinates for the location of the base (usually either the coordinates for the center of the base, or those of the office where the contact person can be reached) – see Fig. 4.6(b).



(a) Mobility Element (b) Location

Figure 4.6: Mobility and Location Elements

- **availability**. This element describes the temporal availability of the base for operations (for instance, one base of operations b_1 may only be available during daytime, but not for nocturnal operations, while another base b_2 may only be available during weekdays, but not during the weekend). If the base is not always available for operations, at least one availability slot must be indicated; each availability slot contains the start and end date and time of the period during which the base is available; also, if the specified availability slot occurs periodically, the rate of recurrence can be specified, as well as the initial and final dates during which the recurrence is valid. Figure 4.7 shows the definition of the availability element and listing 4.1 shows an example of an availability element – in this example, the base would be available every day from January 1 to December 31, 2010, from 9AM to 5PM.

Listing 4.1: Availability Element Example

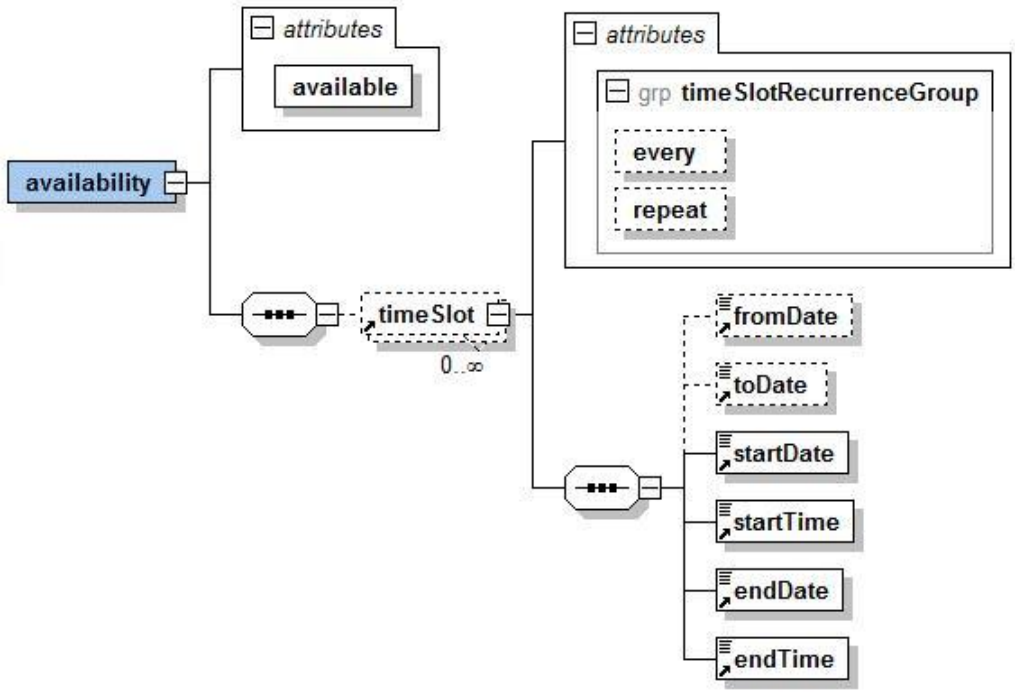


Figure 4.7: Availability Element Definition

```
...
<availability available="periodic">
   <timeSlot repeat="Day" every="1">
      <fromDate>2010-01-01</fromDate>
      <toDate>2010-12-31</toDate>
      <startDate>2010-01-01</startDate>
      <startTime>09:00:00</startTime>
      <endDate>2010-01-01</endDate>
      <endTime>17:00:00</endTime>
   </timeSlot>
</availability>
...
```

- **airport**. This optional element contains a detailed description of the airport within the base of operations, its structure and the services it provides (this element is described in more detail below). The presence of this item is determined by the value of the *air* attribute of the *mobility* element.

- **port**. This optional element contains a detailed description of the port within the base of operations, its structure and the services it provides to boats and/or submarines. The presence of this item is determined by the values of the *water* and *underwater* attributes of the *mobility* element.

- **groundBase**. This optional element contains a detailed description of the ground base within the base of operations, its structure and the services it provides. The presence of

this item is determined by the value of the *land* attribute of the *mobility* element.

Given the complexity of the *airport*, *port* and *groundBase* elements, they are described in detail in the following three sections.

4.2.2 Airport

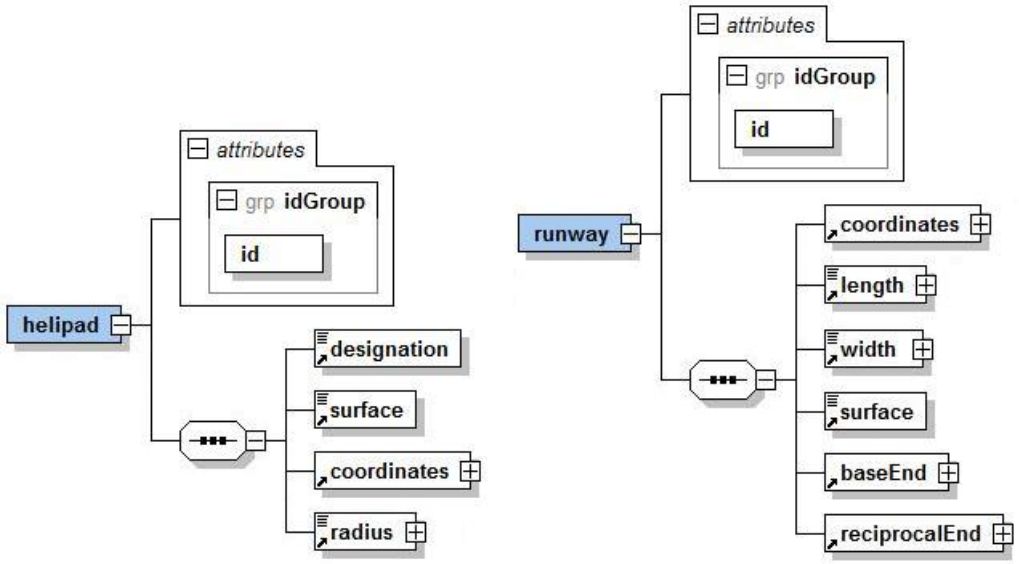
In order to provide a description of airports closer to what is used in the real world, some applications that make use of an airport description were analyzed, as seen in section 2.3. Considering the different formats and information included in the analyzed simulators, it was decided to include a stripped down version of the airport description found in FSX, focusing on the important aspects, such as positioning, dimensions, and intersections of possible paths, leaving out the information regarding visual details, such as lights or signs.

Equation 4.2 shows the contents of the *airport* element, which are explained in more detail below.

$$\begin{aligned} Airport &= \langle name, description, contactPerson, location, \\ &\qquad IATA, ICAO, magVar, H, R, T, P, G, U \rangle \\ H &= \{Helipad\} \\ R &= \{Runway\} \\ T &= \{Taxiway\} \\ P &= \{Parking\} \\ G &= \{Hangar\} \\ U &= \{Utility\} \end{aligned} \tag{4.2}$$

- **name**. The name by which the airport is known.
- **description**. Textual description of the airport and the services it provides.
- **contactPerson**. Information regarding the person of contact for the airport; its definition is the same as for the base of operations.
- **location**. Information regarding the location of the airport; its definition is the same as presented above for the base of operations.
- **IATA**. The IATA (International Air Transport Association) code of the airport; this code is comprised of three letters, and widely used for major airports, namely in baggage tags.
- **ICAO**. The ICAO (International Civil Aviation Organization) code of the airport; this code is comprised of four alphanumeric characters, and provides a unique code for each airport worldwide.

- **magVar**. The magnetic variation (difference, given in degrees, between true North and magnetic North) at the airport location.
- **helipad**. Helipads are relatively small, round or square regions of the airport used by helicopters for vertical takeoff and landing; the helipad element is comprised of four items, as can be seen in Fig. 4.8(a):
  - **designation**. The name by which the helipad is known.
  - **surface**. Material the surface of the helipad is made of.
  - **coordinates**. Coordinates for the center of the helipad.
  - **radius**. Radius of the helipad.



(a) Helipad Element (b) Runway Element

Figure 4.8: Helipad and Runway Elements

- **runway**. Runways are the straight, flat and long strips of terrain used by aircraft for takeoff and landing; the runway element is comprised by several items, as represented graphically in Fig. 4.8(b):
  - **coordinates**. Coordinates for the center of the runway.
  - **length**. Length of the runway.
  - **width**. Width of the runway.
  - **surface**. Material the surface of the runway is made of.
  - **baseEnd**. Contains information regarding one of the two orientations of the runway; it includes the designation of the runway (a number, from 01 to 36, corresponding to one tenth of the magnetic heading of the runway), coordinates for the start and end points of the runway and its orientation (heading).

  - **reciprocalEnd**. Contains information regarding the other orientation of the runway; the designation should differ from the designation of the baseEnd by 18, and the orientation by 180°; the start and end points may match the end and start points of the baseEnd, respectively, but in some cases that may not be the case.

- **taxiway**. Taxiways are used for ground operations (either by aircraft or by other vehicles operating on the airport), connecting runways with other areas of the airport, such as parking spaces, fuel facilities, hangars or helipads; this element is comprised by four items, as shown graphically in Fig. 4.9:

  - **designation**. The name by which the taxiway is known.
  - **surface**. Material the surface of the taxiway is made of.
  - **width**. Width of the taxiway.
  - **path**. Contains information about the shape of the taxiway; it include both initial and final points of the taxiway, as well as a variable number of middle points – where the taxiway changes direction or where it intersects with another taxiway or runway; each point contain the coordinates of its location; in case of interception, the taxiway(s) and/or runway(s) it intercepts with are also specified.

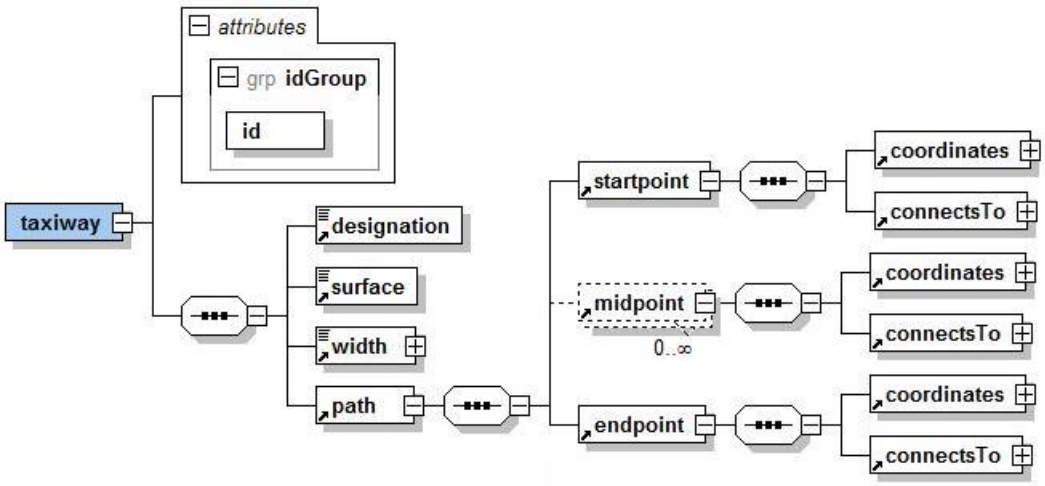


Figure 4.9: Taxiway Element Specification

- **parking**. Parking spaces are specific locations within the airport, used to park aircraft, usually for a relatively short period of time; this element has several items:

  - **designation**. The designation of the parking space.
  - **description**. Description of the parking space, including purposes (whether the parking space is used mainly by commercial aircraft, cargo companies, privately owned jets, or others) and other information.
  - **airlines**. Lists which airlines have priority of use over the specific parking space.

- **coordinates**. Coordinates of the parking space.
- **radius**. Radius of the parking space.
- **connection**. Indicates the taxiway (including the specific coordinates) where the parking space connects to the taxiway network.

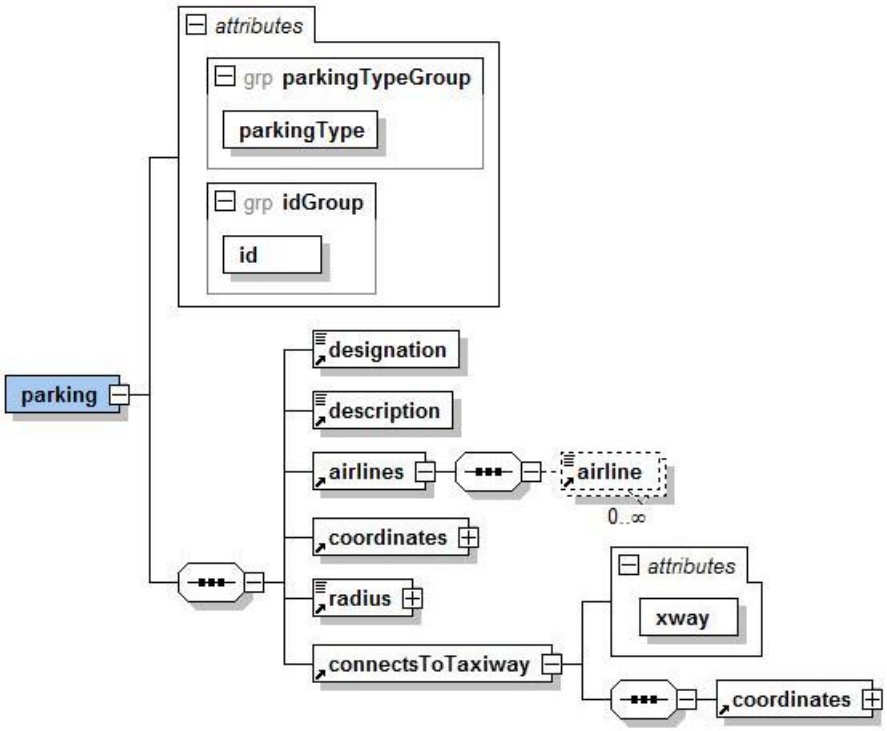


Figure 4.10: Parking Element Specification

- **hangar**. Hangars are closed structures, usually used for long-term housing of aircraft, maintenance or repair operations; this element has three items, as represented graphically in Fig. 4.11:
  - **designation**. The designation of the hangar.
  - **description**. Brief description of the hangar, especially in terms of its purposes and possible owner.
  - **shape**. Specifies the shape of the hangar (expressed as a polygon), its height, useful area and the type, location and size of the doors.
- **utility**. There are four types of utilities – Tower, Fuel Facility, Battery Facility and Water Facility – with three common elements:
  - **designation**. The designation of the utility.
  - **coordinates**. Coordinates for the center of the utility.
  - **radius**. Radius of the utility.

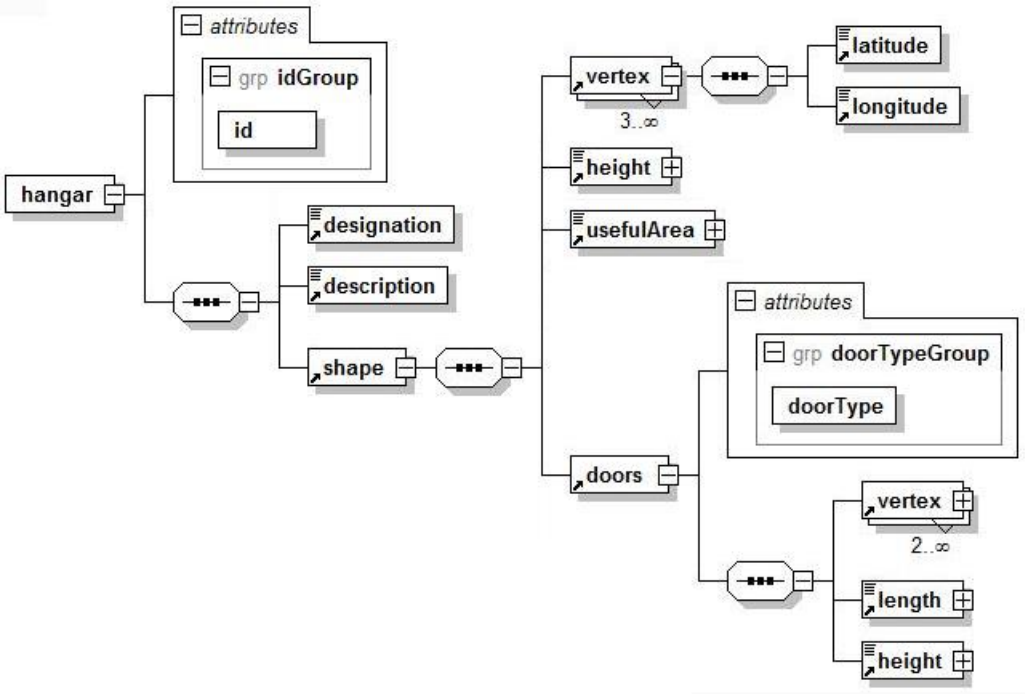


Figure 4.11: Hangar Element Specification

In addition to these three elements, the Tower also has an element specifying its height; both Fuel and Water Facilities have the available quantity of fuel or water, respectively; and the Fuel Facility has an indication of the type of fuel it provides.

The information contained in these element (namely, the information retrieved from the *runways* and *taxiways* elements) is structured in a manner that facilitates its use in the construction of a graph containing the possible paths to be used by airplanes while on the ground [Sousa, 2010]. This information is of major importance so that the agents can plan their paths with maximum efficiency.

4.2.3 Port

The port has a similar structure to the airport, but adapted to meet the requirements for modeling a water facility. Figure 4.12 represents the port element graphically, and the several elements that constitute a port are enumerated below:

- **name**. The name by which the port is known
- **description**. Brief description of the port and its facilities and the services it provides
- **contactPerson**. The details for the person who should be contacted for matters regarding the port. The definition of this element is the same as the one presented above

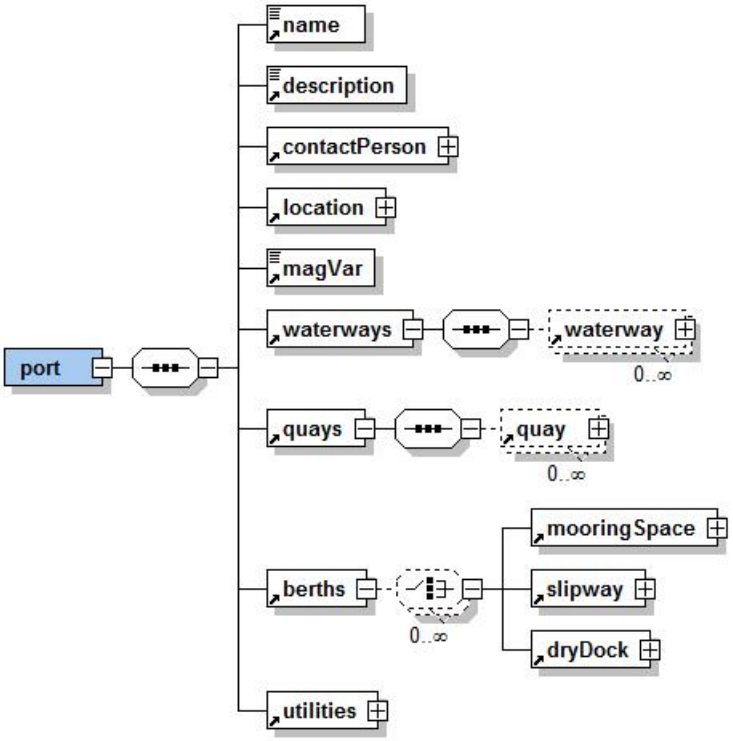
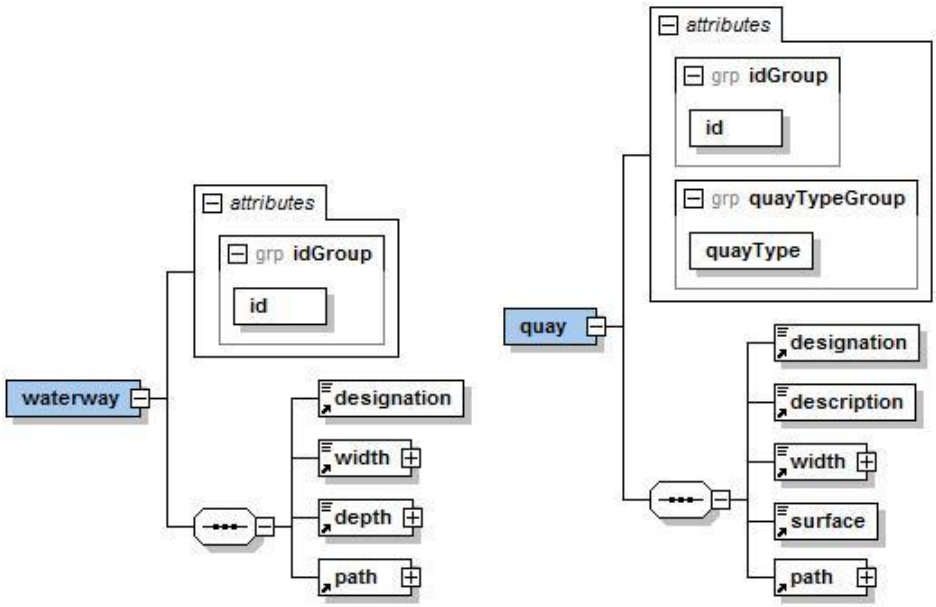


Figure 4.12: Port Element Specification

- **location**. The location of the port (the definition of this element is the same as presented above)
- **magVar**. The magnetic variation at the location of the port
- **waterways**. This element describes the virtual roads of water that can be used by boats and submarines. Each waterway has a unique identifier and the following elements, as represented graphically in Fig. 4.13(a):
  - **designation**. The designation by which the waterway is known
  - **width**. The width of the waterway, which also represents the maximum width of the vessels that can navigate through that waterway
  - **depth**. The depth of the waterway, which also limits the vessels that can use the waterway
  - **path**. of the waterway; again, and just as in the taxiway element, the path is described by a start point, an endpoint and a variable number of midpoints in between them; the order in which the path elements appear specifies the default traffic direction along the waterway
- **quays**. This element describes the support structures that exist by the water (or even penetrating the body of water), and that are usually used for accessing the places where boats



(a) Waterway Element (b) Quay Element

Figure 4.13: Waterway and Quay Elements

are 'parked'. Each quay has a unique identifier and the following elements, as can be seen in Fig. 4.13(b):

- **quayType**. An indication of the type of quay, which can be chosen among pier, jetty or mole
- **designation**. The designation by which the quay is known
- **description**. A brief description of the quay (some historical aspects can be included here, for instance)
- **surface**. The material in which the quay is built
- **width**. The width of the quay
- **path**. The path of the quay; once again, the path is expressed with the start, mid and endpoints, each point having the possibility to connect to a point in another quay

- **berths**. This element contains all locations that can be used to park a boat or submarine. There are three distinct kinds of berths - mooring spaces, slipways and dry docks. There are four common elements to these three types of berths:
  - **designation**. The designation by which this berth is known
  - **description**. A brief description of the berth and it main characteristics
  - **boatType**. This element contains a number of boolean attributes that specify the types of vessels that can use the specific berth
  - **coordinates**. The coordinates for the location of the berth

Each of the three types also has specific elements, detailed below:

- **mooringSpace**. Mooring spaces are specific locations within the port where a boat can be moored (a boat is said to be moored when it is secured to a fixed structure, usually by ropes). A mooring space is comprised of a number of elements:
  - **depth**. Specifies the depth at the mooring space location, which limits the boats that can only use the specific mooring space
  - **maxBoatLength**. This element specifies the maximum length a boat can have to use the specific mooring space
  - **mooringWidth**. Specifies the width of the mooring space, which also specifies the maximum width of the boats that can moore at that location
  - **mooresTo**. This element indicates which quay is used for mooring, and which side of the boat should face the quay - port, starboard, bow or stern
- **slipway**. Slipways are ramps used to load and unload boats to and from the water, usually using either vehicles with a trailer that can transport water vehicles, or using the natural tides and other mechanical apparatus. Slipways can also be used for the construction of new boats, or for repairs to be conducted. A slipway has the following additional fields:
  - **length**. Specifies the length of the slipway
  - **width**. Represents the width of the slipway
  - **angle**. Indicates the angle between the slipway surface and the horizontal plane
  - **surface**. Specifies the material the slipway is built of
  - **maxWeight**. Indicates the maximum weight supported by the slipway (which constraints the vessels that can use the specific slipway)
- **dryDock**. Dry docks are closed structures used for the construction or maintenance operations of vessels. A dry dock can be flooded for the vessel to enter or leave; once the boat is inside and the doors closed, it can be drained, and the vessel laid to rest on a solid support structure, so that operations can be made in a water-free environment. A dry dock has the following additional fields:
  - **length**. Specifies the length of the slipway
  - **width**. Represents the width of the slipway
  - **depth**. Indicates the depth of the dry dock, when opened to the water
  - **height**. Indicates the maximum height of the dock
  - **dryDockVolume**. Specifies the volume of water the dry dock can hold
  - **waterFlow**. Specifies the rate at which water is pumped in or out of the dock. Together with the volume of the dock, this can also be used to determine the amount of time it takes for the dock to fill or to drain.

- **utilities**. This element has the same structure as the one presented above for the airport.

4.2.4 Ground Base

The ground base is intended to provide support for ground vehicles, and is defined as a structure similar to the one that describes an airport, or a port. Figure 4.14 represents the *groundBase* element graphically, and the several elements that constitute a ground base are enumerated below:

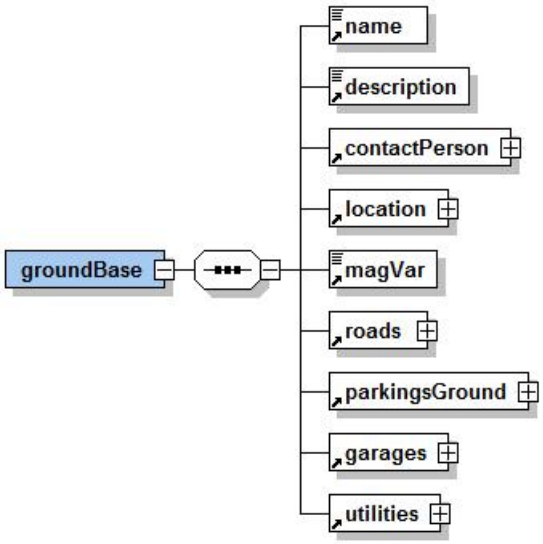


Figure 4.14: *groundBase* Element Specification

- **name**. The name by which the ground base is known
- **description**. Brief description of the ground base and its facilities and the services it provides
- **contactPerson**. The details for the person who should be contacted for matters regarding the ground base; the definition of this element is the same as the one presented above
- **location**. The location of the ground base (the definition of this element is the same as presented above)
- **magVar**. The magnetic variation at the location of the ground base
- **roads**. Describes the network of roads that exists within the ground base. It is comprised by any number of *road* elements, each of which represents one road. Each *road* has a unique identifier and the following elements:
  - **name**. This optional attribute represents the name by which a road is known, in the case it possesses an official designation and a common name
  - **designation**. This element represents the official designation of the road
  - **surface**. Indicates the material that paves the road
  - **width**. The total width of the road (all lanes are considered in this measurement)

  - **totalNumberOfLanes**. This element indicates the total number of lanes in the road, and includes an attribute named **reverseLanes** that indicates how many of those lanes are used for traffic in the reverse direction of the road specification (the direction is based on the path element, presented below); the number of reverse lanes must not exceed the total number of lanes ($0 \leq reverseLanes \leq totalNumberOfLanes$)
  - **trafficIsRightHand**. This boolean element indicates if the traffic is right-handed (as is the case in most countries), or left-handed (as is the case of the United Kingdom, Australia, India and several south-African countries)
  - **path**. This element specifies the geometry of the road; its definition is the same as already presented above

- **parkingsGround**. Describes the existing parking lots and individual parking spaces within each parking lot; each parking lot has a unique identifier and:
  - **designation**. The name by which the parking lot is known (for instance P1 or P2)
  - **description**. A description for the parking lot (this field can be used to describe if a parking lot is meant to be used solely by base personnel, or by visitors, for instance)
  - **parkingSpace**. A parking lot may contain any number of parking spaces, each of which has a unique identifier, the coordinates and size of the parking space and also an indication of the type of space (it may specify, for instance, if the space is meant for heavy of light vehicles, or if it can only be used by electric vehicles)

- **garages**. Describes the structures that can be used by the ground vehicles as garages, either for repairs and maintenance, or even for parking during the night or during adverse weather conditions; each garage has a unique identifier, designation (the name of the garage), description (it can be used to describe the specific garage and the services it provides) and the description of the shape of the garage (the shape follows the same definition as for hangars, as presented above)

- **utilities**. This element has the same structure as the one presented above for the airport

It was mentioned in the description of taxiways, waterways, quays and roads that each individual element can be connected to others of the same type, as to construct the taxiway network for the airport, the waterway and quay networks for the port, and the road network for ground bases. Additionally, taxiways, roads, quays and waterways can also contain connections to each other, as to provide an interconnected network of transportation lines, so that vehicles can travel from one point of a base of operations to another. For instance, an autonomous ground vehicle with a trailer that carries a boat can travel from the ground base by road and connect to the quay network at the port, in order to launch the boat to the water using a slipway. Also, by providing a connection

between the several water and land networks, it provides support for amphibious vehicles to use all the resources available at the base of operations.

4.2.5 No Fly Areas

This section of the scenario description file has a straightforward purpose – clearly defining all areas that cannot be navigated through, at a global level, by any vehicle of any team. Even though the name suggests that the areas defined in this section only apply to airspace and air traffic, the areas can be defined for air, land, water and/or underwater vehicles.

Every *area* element has a unique identifier, and several elements, as defined in Table 4.3, and shown summarily in Equation 4.3.

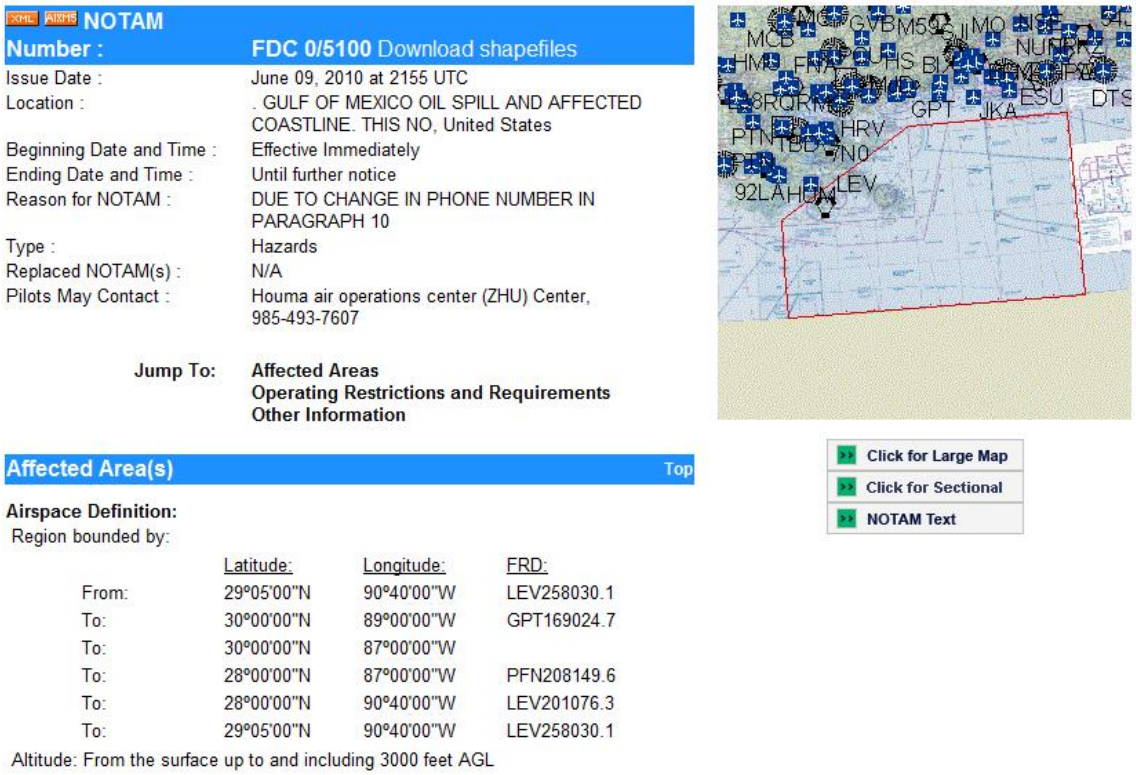
Element	Description
denomination	The name that identifies the area
medium	The medium the prohibition applies to; this element has the same definition as the one presented for the base of operations, with four boolean attributes – land, air, water and underwater–, which determines the vehicles that cannot navigate through the area
availability	Specifies the temporal availability (or unavailability, in this case) of the area (for instance, aircrafts may not fly over a given area n_1 at night, but they may do so during daytime); the definition of availability is also the same as the one shown above for the base of operations
shape	The shape of the area can be expressed as either a polygon or a circle, extruded vertically from minimum to maximum altitudes; a polygon is comprised of three or more vertexes, each identified by latitude and longitude, while a circle is comprised of the center coordinates (latitude and longitude) and a radius

Table 4.3: Definition of the Area Element

$$
\begin{aligned}
&Area_{ID} = \langle Denomination, Medium, Availability, Shape \rangle \\
&Shape = Polygon \vee Circle \\
&Polygon = \left\langle minAlt, maxAlt, \{vertex\}^{3+} \right\rangle \\
&Circle = \langle minAlt, maxAlt, center, radius \rangle
\end{aligned}
\tag{4.3}
$$

This definition of area can be extended in the future to include other geometrical forms, but this definition corresponds to how these areas are usually defined in the real world – for instance, the FAA defines the areas for Temporary Flight Restrictions (TFRs) using this definition [FAA, 2010b]. Figure 4.15 shows an example of an FAA-issued flight restriction over the Gulf of Mexico; as can be seen in both the airspace definition and the area map, the restricted area is defined by a polygon, each vertex defined by latitude and longitude; the altitude of the area is defined by minimum and maximum altitudes (in this case, from 0 to 3000 feet above ground level). If

the restriction applies to water and/or underwater vehicles, the altitude will have a negative value, which is interpreted as the depth of the area.



NOTAM

Number :	**FDC 0/5100** Download shapefiles
Issue Date :	June 09, 2010 at 2155 UTC
Location :	. GULF OF MEXICO OIL SPILL AND AFFECTED COASTLINE. THIS NO, United States
Beginning Date and Time :	Effective Immediately
Ending Date and Time :	Until further notice
Reason for NOTAM :	DUE TO CHANGE IN PHONE NUMBER IN PARAGRAPH 10
Type :	Hazards
Replaced NOTAM(s) :	N/A
Pilots May Contact :	Houma air operations center (ZHU) Center, 985-493-7607

Jump To: **Affected Areas** **Operating Restrictions and Requirements** **Other Information**

Affected Area(s) Top

Airspace Definition:
Region bounded by:

	Latitude:	Longitude:	FRD:
From:	29°05'00"N	90°40'00"W	LEV258030.1
To:	30°00'00"N	89°00'00"W	GPT169024.7
To:	30°00'00"N	87°00'00"W	
To:	28°00'00"N	87°00'00"W	PFN208149.6
To:	28°00'00"N	90°40'00"W	LEV201076.3
To:	29°05'00"N	90°40'00"W	LEV258030.1

Altitude: From the surface up to and including 3000 feet AGL

Figure 4.15: Example of a FAA Temporary Flight Restriction [3]

4.2.6 Controllers

This section of the scenario description file contains a list of traffic controllers (for air, water or land traffic), and their respective details – a unique identifier for each controller; the base of operations $b \in B$ it is associated with; the level of autonomy a vehicle has when moving through the controller area, indicated by the *requiredAction* attribute; an indication of the type (*role*) of controller it represents (for the moment, only civilian and military roles are considered for this field); the area it has jurisdiction over (area definition is the same as shown above); and the contact frequencies (for instance, approach and departure control can be handled in a different frequency than ground control). Figure 4.16 shows the information contained within this element in a graphical notation.

This information is used by ATC Agents as well as by the vehicle agents to determine their behavior when navigating within the areas defined in these elements. For instance, one Controller c_1 may have full control over the area surrounding the airport (as air traffic controllers for major airports do), and all agents representing vehicles moving through that area would have to get approval from the controller for every decision; another Controller c_2 may have more of a passive behavior (as do some controllers of small airfields), simply being informed of the vehicle

[3] Image retrieved from `http://tfr.faa.gov/save_pages/detail_0_5100.html`, in October 2010

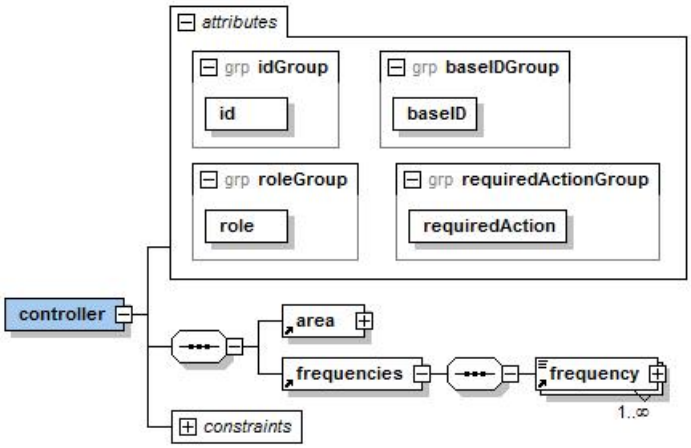


Figure 4.16: Controller Element Definition

decisions, but having no direct control over them. There are three levels of control defined in the *requiredAction* attribute: in the lowest level (inform), the controller is only informed of the vehicle decisions, and has no control over them; in the intermediary level (request), the vehicle has the autonomy to make decisions, but not to implement them without the consent of the controller; in the highest level (obey), vehicles have no autonomy regarding motion, and have to obey to all decisions made by the controller and sent to them.

4.2.7 Agent Types

This section of the scenario description file contains a list of available vehicle types. Each vehicle type has a unique identifier, which will later be referenced by the TDL file (see section 4.3.1) to indicate the type of each vehicle that composes the team. The information regarding the vehicle type is divided into five categories:

- **Simulated agent**. This category contains only one element – title. The title is a string that uniquely identifies a vehicle type within the simulation platform; several details regarding the vehicle type can be retrieved from the simulator, given this string.
- **Real agent**. Information describing the real vehicle includes the category of the vehicle (aircraft, car, boat or submarine), type, manufacturer, model and variation, as well as any additional informational details one might want to include.
- **Physical**. This category includes characteristics regarding vehicle dimensions as well as other physical aspects. Several elements are common to all types of vehicles (such as cargo or fuel capacity), but some elements are dependent on the vehicle type (for instance, only aircraft have wingspan or wing area). Table 4.4 lists all the elements in this category common to all vehicle types, and Table 4.5 lists the elements that are specific to each vehicle type.

Element	Description
emptyWeight	Weight of the vehicle, when empty (without fuel or cargo)
maxPayload	The maximum weight of the cargo that can be loaded onto the vehicle
maxFuel or maxBattery	Indicates the maximum amount of fuel the vehicle can be fueled with; in case the vehicle is completely moved on electrical power, the battery capacity is indicated
length	Length of the vehicle
height	Height of the vehicle, measured at the highest point (in planes, this is usually the tail)
nEngines	Number of engines of the vehicle; this element includes an attribute that indicates the type of engine

Table 4.4: Agent Type Common Physical Elements

Vehicle Type	Element	Description
Aircraft	wingspan	The width of the aircraft, measured from one wingtip to the other
	wingarea	The area occupied by the wings of the aircraft, which will provide lift
Cars	width	The width of the car
	nWheels	The total number of wheels of the vehicle; this element also includes an attribute that specifies how many of those wheels have traction
Boats and Submarines	beam	The width of the vessel
	rudderArea	This optional element indicates the area of the rudder
	anchorLength	This optional element indicates the length of the anchor; an additional attribute, named *nAnchors*, indicates how many anchors the vessel has
	mooringLines	This optional element indicates the total length of the vessel's mooring lines; an additional attribute, named *nLines*, indicates how many mooring lines the vessel has
Boats	maxDraft	The maximum depth of the boat, as measured from the water line, when fully loaded (this value will have an impact when choosing the waterways the boat can navigate through, for instance)
Submarines	seaplaneArea	This optional element indicates the area occupied by the seaplanes
	waterTanks	This optional element indicates the total volume of water that can be held in the submarine's ballast tanks; this element also includes an attribute, *nTanks*, that specifies how many ballast tanks the submarine has

Table 4.5: Agent Type Specific Physical Elements

- **Performance**. Performance information includes elements regarding the operational performance of the vehicle. Several of these elements are common to all vehicle types, as shown in Table 4.6, while others are specific to the vehicle type. Table 4.7 shows the elements specific to each vehicle type

Element	Description
cruiseSpeed	The cruise speed of the vehicle is the usual speed at which the vehicle operates
maxSpeed	The maximum speed of the vehicle is the speed at which it can travel, without the risk of structural damages
range	This element indicates the maximum operational range of the vehicle, without considering refueling operations
fuelFlow or energyFlow	This element indicates the average fuel consumption of the vehicle (considering it is traveling at cruise speed, and, in the case of aircraft, a straight and level flight[4])
dragCoefficient	This element measures the resistance of the vehicle to motion

Table 4.6: Common Agent Type Performance Elements

Vehicle Type	Element	Description
Aircraft	maxTakeoffWeight	The maximum weight of the aircraft during takeoff
	requiredRunwayLength	The minimum length of a runway so that the aircraft can safely takeoff and land
	stallSpeed	The speed under which the aircraft will no longer be able to generate enough lift to sustain flight
	climbRate	The maximum vertical velocity of the aircraft, when climbing
	ceiling	The maximum altitude at which the aircraft can operate
Cars	maxAttackAngle	The maximum terrain inclination angle the vehicle is able to climb
	min90DegTurnRadius	The radius of the circle traveled by the vehicle when performing a turn; if the vehicle can rotate over its own axis, the radius will be 0
Boats	min90DegTurnRadius	The same definition as above
Submarines	min90DegTurnRadius	The same definition as above
	maxUnderwaterSpeed	The maximum speed the submarine can travel at, when submerged
	maxDepth	The maximum depth at which the submarine can navigate and operate, without the risk of structural damages
	maxVerticalVelocity	The maximum vertical velocity of the submarine

Table 4.7: Specific Agent Type Performance Elements

- **Payload layout**. Payload layout includes information regarding each payload station, specifically its location (in relation to the aircraft's center), dimensions and maximum cargo it can transport. This information will later be used in the team definition process as restrictions to the cargo or sensors each payload may contain – see section 4.3.2.

Figure 4.17 shows the definition of the *agentType* element in a graphical notation.

[4] An aircraft is considered to be flying a straight and level flight when it maintains both altitude and heading, thus describing a straight line.

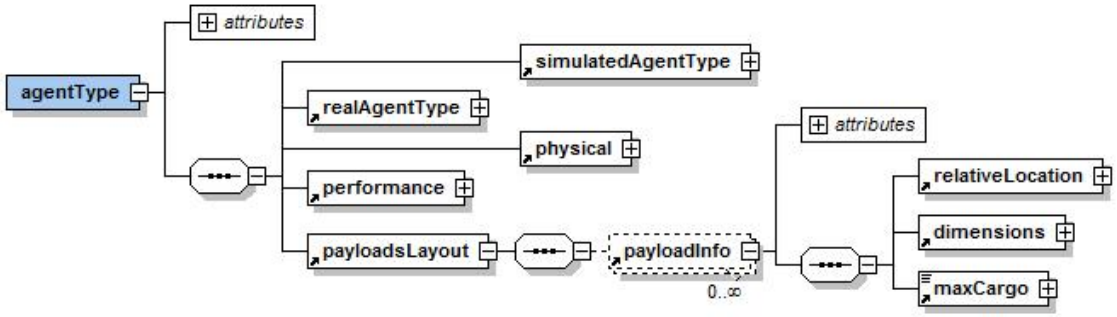


Figure 4.17: Agent Type Element Definition

4.3 Team Description Language

In this section, an overview of TDL is given. The TDL file allows for the description of any number of teams. Each team has a unique identifier and contains several elements, such as the bases of operations $U = \{u_1, u_2, ..., u_{nu}\} \in B$ it can use, additional no fly areas $A = \{a_1, a_2, ..., a_{na}\}$, and the description of all vehicles $V = \{v_1, v_2, ..., v_{nv}\}$ composing the team – see Eq. 4.4.

$$\begin{aligned} &Team_{ID} = \langle Name, Description, History, Purposes, Mobility, ContactPerson, A, U, V \rangle \\ &A = \{Area\} \\ &U = \{UsableBaseOfOperations\} \\ &V = \{Agent\} \end{aligned} \tag{4.4}$$

It is important to notice that TDL is intended to solely specify team composition and operational constraints; all other information, namely related to team organization, hierarchy, behavior, and others, are specified at the mission level, using MDL.

Table 4.8 describes the contents of the *team* element in more detail.

Element	Description
name	The name of the team
description	Textual description of the team and its capabilities
history	Description of the team's history; it may include information regarding past missions, team composition through time or any other information
purposes	Describes the purposes of the team, such as the type of missions it was designed to perform, or is capable of performing
mobility	Indicates the aggregated mobility of the team (the definition of this element is the same as presented above)
contactPerson	Information regarding the person of contact for the team; the definition of this element is the same as for the base of operations
additionalNoFlyAreas	This element contains a list of *area* elements, with the same definition as shown above (see section 4.2.5), that indicates additional areas the specified team cannot navigate through (even though other teams might be able to). As such, the specific team cannot navigate through both the areas specified in SDL and in TDL, $N \cup A$

Element	Description
usableBaseOfOperations	This element contains a list of references to the identifier of the base of operations, as defined in the SDL file (see section 4.2.1), and indicates which bases can be used by the team
agent	Contains a description of each vehicle that composes the team. This element is detailed below

Table 4.8: Team Element Definition

4.3.1 Agents

This section of the team description file contains the key element of the team – the list of vehicles that compose the team, each containing the following information, in addition to a unique identifier, and as graphically shown in Figure 4.18:

- **agentTypeID**. This attribute $agentTypeID \in T$ references the vehicle type identifier, as defined in section 4.2.7
- **name**. The name by which the vehicle is known within the team
- **description**. A textual description of the vehicle, and its foremost capabilities
- **initialLocation**. The initial location of an aircraft is given by the identifier $b \in U$ of a base of operations, the identifier of a parking space, hangar, helipad or utility location within the airport of the base of operations b, and the direction the aircraft is facing. For boats or submarines, the location will refer a specific berth or utility within the port of the base of operations. For cars, the location will refer to a parking space, a garage or utility within the ground base of the base of operations
- **state**. Vehicle status includes information regarding the amount of fuel the vehicle has (or the battery level, in case the vehicle is electric), as well as several elements depending on the vehicle type, as can be seen in Table 4.9

Vehicle Type	Element	Description
Aircraft	lights	This element has a set of boolean attributes that indicate which lights of the aircraft are turned on. Aircraft lights include cabin, logo, wing, recognition, panel, strobe, taxi, landing, beacon and nav.
	doorsOpen	This boolean element indicates whether or not the doors are opened
	gearDown	This boolean element indicates whether or not the landing gear is down
	flapsHandlePosition	This element represents the position of the flaps handle (most aircrafts have three to four possible positions for the flaps)
	rudder	Represents the angle of the rudder surface in respect to the plane's longitudinal axis

Vehicle Type	Element	Description
Aircraft	aileron	Represents the angle of the aileron surfaces, considering a right roll effect as the positive value
	elevator	Represents the angle of the elevator surface, in respect to the aircraft's horizontal plane
Cars	lights	This element has a set of boolean attributes that indicate which lights of the car are turned on. Car lights include cabin, frontal and rear left and right lights
	doorsOpen	Indicates whether or not the doors are opened
	equivalentWheelTurnAngle	Represents the equivalent (combined) turn angle of the wheels, in respect to the front of the car.
Boats	lights	This element has a set of boolean attributes that indicate which lights of the boat are turned on. Boat lights include cabin, top of the mast, stern, side lights, towing, all around light and strobe
	anchor	This optional element indicates how much anchor line is being used; if the boat has an anchor but is not anchored, this value will be 0
	rudder	This optional element represents the angle of the rudder surface in respect to the boat's longitudinal axis
	equivalentEngineAngle	Represents the equivalent (combined) engine turn angle, in respect to the rear of the boat (the direction in which the thrust will occur)
Submarines	lights	This element has a set of boolean attributes that indicate which lights of the submarine are turned on. Submarine lights include cabin, rudder, port, starboard, bow and strobe
	anchor	Same definition as above
	rudder	Same definition as above
	seaplane	This optional element represents the angle of the seaplane surface in respect to the submarine's horizontal plane
	equivalentEngineAngle	Represents the equivalent (combined) engine turn angle, along the horizontal plane, in respect to the rear of the boat (the direction in which the thrust will occur)
	equivalentEnginePitch	Represents the equivalent (combined) engine turn angle, along the vertical plane, in respect to the rear of the boat (the direction in which the thrust will occur)

Table 4.9: Specific Agent State Elements

- **realAgent**. Real vehicle information includes communication parameters and ATC-related information. Communication parameters include the vehicle control frequency, as well as the active and standby communication frequencies (when an actual vehicle is being used, these values are also used to configure the external module wrapper with the frequencies used to communicate with the vehicle). ATC information includes a call sign (the name by

which the vehicle responds), as well as some information that varies according to vehicle type, as shown in Table 4.10

Vehicle Type	Element	Description
Aircraft	tailNumber	The tail number is the registration code of the aircraft, the equivalent to a license plate
	name	Name of the aircraft
Cars	VIN	The Vehicle Identification Number of the ground vehicle
	plate	The license plate
Boats and Submarines	hullNumberIMO	The hull number or International Maritime Organization (IMO) registration number for the vessel
	name	Name of the vessel

Table 4.10: Specific Real Agent Registration Elements

- **simulatedAgent**. Simulated vehicle information provides information regarding the simulated counterpart of the vehicle, and includes the vehicle's tail number (which is necessary to instantiate the vehicle in the simulation platform), which should be the same as the one specified for the real vehicle, and may not be repeated within the simulator.
- **payload**. Specifies the contents of each payload station, as detailed below

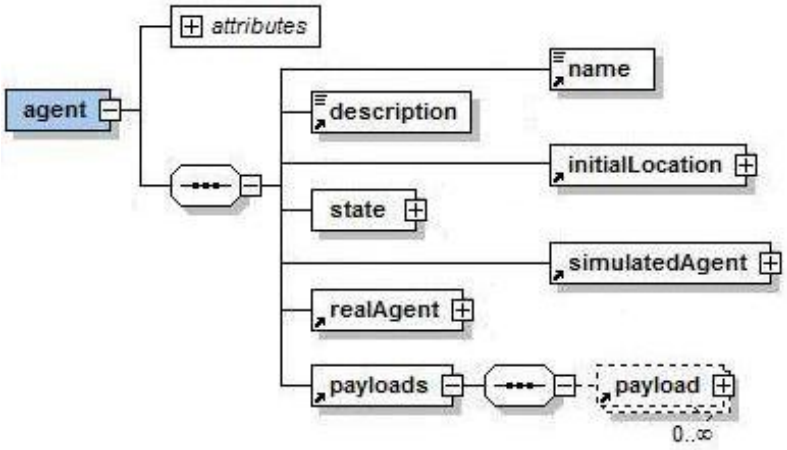


Figure 4.18: Agent Element Definition

4.3.2 Payloads

For each payload station identified in the vehicle type $agentTypeID \in T$ definition (see section 4.2.7), the details about the sensors it carries and/or cargo it transports are provided, as can be seen in Fig. 4.19.

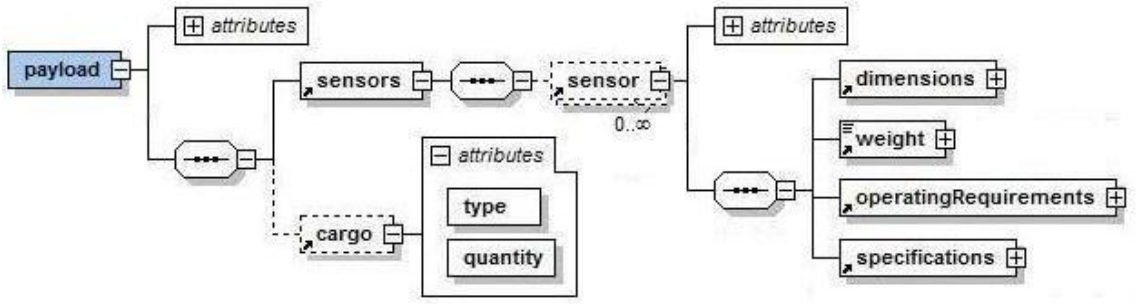


Figure 4.19: Payload Element Definition

Sensor definition includes the sensor type (temperature sensor, carbon dioxide (CO_2) detector, carbon monoxide (CO) detector, infrared or visible wavelength spectrum camera, and so on), its dimensions (specified as the dimensions for the three-dimensional bounding box for the sensor equipment), weight, operational requirements (details such as temperature or humidity ranges, voltage and current specifications, or power consumption can be specified in this element) and any other specifications details one might want to add (such as the accuracy of the sensor, its response time, the range of detected values, or the output definition).

The payload station can also contain cargo, in which case the type and quantity of such cargo are indicated (for instance, 350 liters of fire retardant).

4.4 Disturbance Description Language

In this section, a full overview of the Disturbance Description Language (DDL) is given, and each of its elements detailed. As previously stated, this dialect is intended to describe any element within the environment that can be considered out of the ordinary, and that, for that reason, requires some sort of action from the team of vehicles. The root element of a DDL file is the *disturbances* element, which may contain any number of disturbances, $D = \{d_1, d_2, \cdots, d_n\}$. Each disturbance has a unique identifier, an indication of the type of disturbance being described (including fire, person, vehicle, hydrothermal vent and pollution focus) and a designation, as well as location, availability, mobility and component configuration. Each of the latter four elements is the focus of a more in-depth analysis below. Equation 4.5 shows the root element of the DDL file, *disturbances*, and its definition, and Fig. 4.20 shows the definition of *disturbances* in a graphical notation.

$$\begin{aligned} &Disturbances = \{Disturbance\} \\ &Disturbance = \left\langle id, type, denomination, Location, Availability, Mobility, \{Component\}^{1+} \right\rangle \end{aligned} \tag{4.5}$$

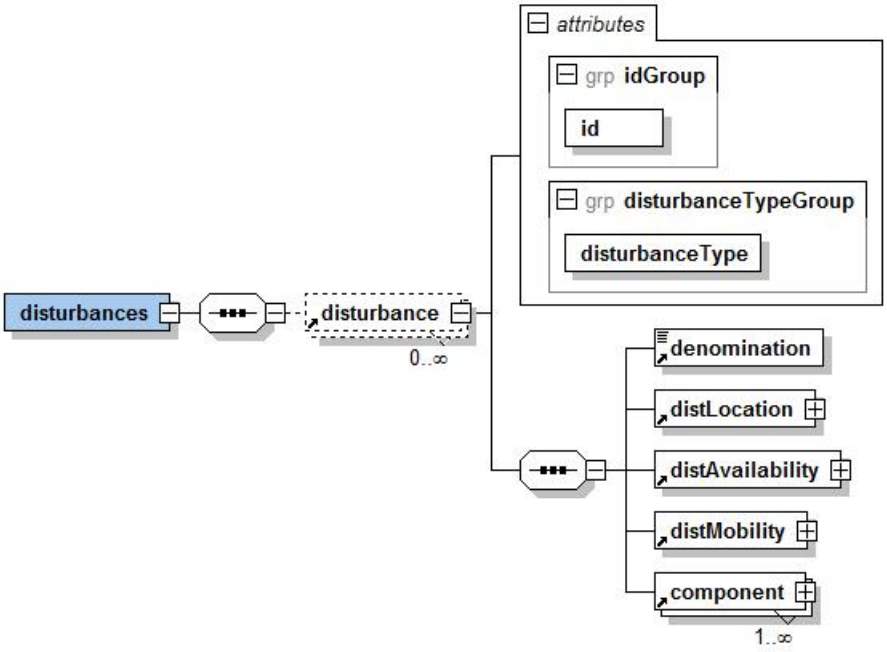


Figure 4.20: Disturbances Element Definition

4.4.1 Location

Location specifies the initial physical placing of the disturbance. Disturbances can be created in a specific or at a random location. In the first case, the coordinates of the initial location must be provided (by specifying the latitude, longitude and altitude of the location); in the second case, an area must be specified, so that the disturbance can be created at a random location within that area. The definition of the *area* element is the same as presented above, in section 4.2.5.

4.4.2 Availability

Availability specifies the temporal existence of the disturbance. Disturbances can have a scheduled or random start. In the first case, a specific point in time is indicated; this can be defined as either a *timestamp*, or a duration (a temporal offset, counting from simulation start). In the case of a random start, a time period is indicated by two values – start and end times (again, using either absolute *timestamps* or durations, counting from simulation start).

The end of a disturbance can be scheduled to occur at a specific point in time, at any point during a specified time interval (random), or it can be unspecified. In the first two cases (scheduled or random end), the definition follows the same rules as for the disturbance start. In the last case, the disturbance may not have an end (as in the case of hydrothermal vents, for instance), or it can be determined by other factors, such as mobility or size growth pattern. The end of a disturbance may also be dependent on other external factors, such as the weather or the actions of the team of vehicles. For instance, in the case of a fire, it ends only when extinguished, either by environmental factors, such as rain, or by the actions of the vehicles within the environment.

Figure 4.21 shows the *distAvailability* element in a graphical notation.

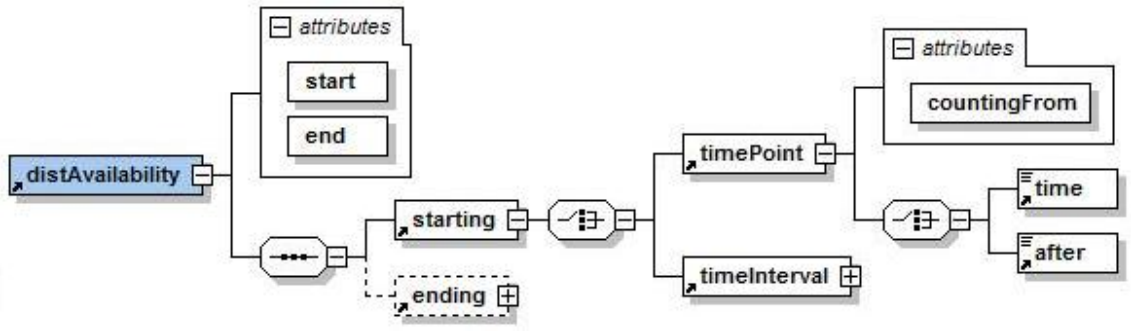


Figure 4.21: Disturbance Availability

4.4.3 Mobility

Mobility describes how a disturbance moves through the scenery over time. Considered mobility patterns include general heading, random motion, specified path following and motion according to the wind (in addition to stationary disturbances, which do not move over time). In each of the cases, a different set of parameters are specified, as can be seen in Fig. 4.22:

- **General heading**. In case of a general heading mobility type, the desired heading must be specified, as well as a randomness factor, specified by the maximum heading variation and the distribution of the variation (considered distributions include uniform, triangular or normal). Heading variation is measured in degrees, and is applied to both the right and left sides of the defined general heading. For instance, a disturbance can be specified as moving West (270°), with a uniform variation of 15°, which means that it moves in the $[255-285]^{\circ}$ heading interval.
- **Random**. In case of a random motion pattern, the maximum heading variation must be indicated as well as the time period it applies to (for instance, a maximum variation of 30° per hour can be specified).
- **Path**. If a predefined path should be followed by the disturbance (as can be the case of a vehicle or person), the path must be specified as an ordered set of coordinates.
- **Wind**. If the disturbance moves in accordance to the wind, there is no need for additional parameters, as motion will be determined by the simulation platform according to environmental conditions.

If the disturbance is not stationary, the speed at which it moves must also be specified in addition to the motion pattern and respective parameters. Speed can be constant, or it can vary according to a number of factors, such as time or location. For simplification purposes, a number of predetermined generic functions (including linear, polynomial, exponential, logarithmic, sinusoidal, and others) are available for customization, through a series of coefficients, to describe

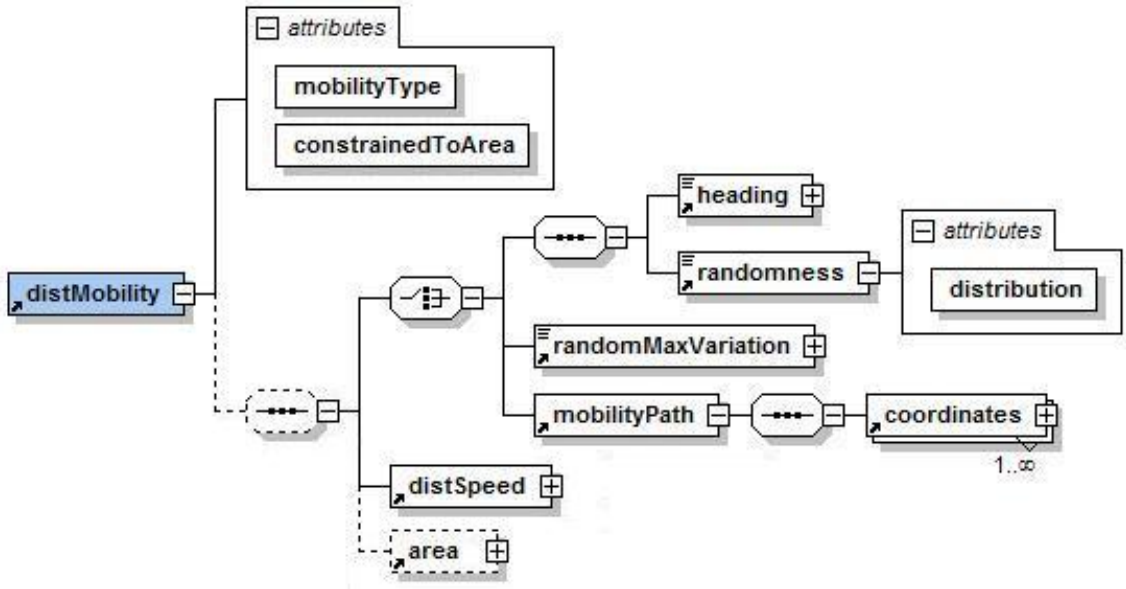


Figure 4.22: Disturbance Mobility

how speed varies, as shown in Table 4.11. In the four last examples, C_1 and C_2 control the vertical and horizontal displacement of the function, respectively, while C_3 and C_4 control the horizontal and vertical stretching of the function. V represents the variable the function depends on (such as time, latitude, longitude, altitude, among others). Also, the speed can be defined by branches, each branch delimited by values of V, and defined by an independent function.

A disturbance may also be limited to a given area, in which case the area is also specified (the *area* element definition is the same as presented above in section 4.4.1).

One point to have in consideration is the relation between speed and size (detailed below). For instance, in the case of a fire, it usually moves according to the direction of the wind (in addition to factors such as soil composition, density and flammability, terrain elevation profile, and many others). However, since the fire also grows, the center of the fire, which is the point considered in motion, moves only about half of the distance of the front of the fire; this has to be factored in, when specifying the coefficients for motion speed.

Motion Speed Pattern	Definition
linear	$C_2V + C_1$
polynomial	$C_nV^{n-1} + \cdots + C_3V^2 + C_2V + C_1$
exponential	$C_4^{C_3V+C_2} + C_1$
logarithmic	$log_{C_4}(C_3V + C_2) + C_1$
root	$C_5 \sqrt[C_4]{C_3V + C_2} + C_1$
sinusoidal	$C_4 \cos(C_3V + C_2) + C_1$

Table 4.11: Example Disturbance Speed Patterns

4.4.4 Components

This section of the DDL file contains the various possible components of a disturbance, and their unique characteristics. These components represent different aspects of a disturbance – for instance, a fire can have a visual component, a temperature component and a CO_2 component, each with it's own size, growth or dispersion patterns, and each requiring distinct sensors to be detected. Figure 4.23 shows the definition of the *component* element.

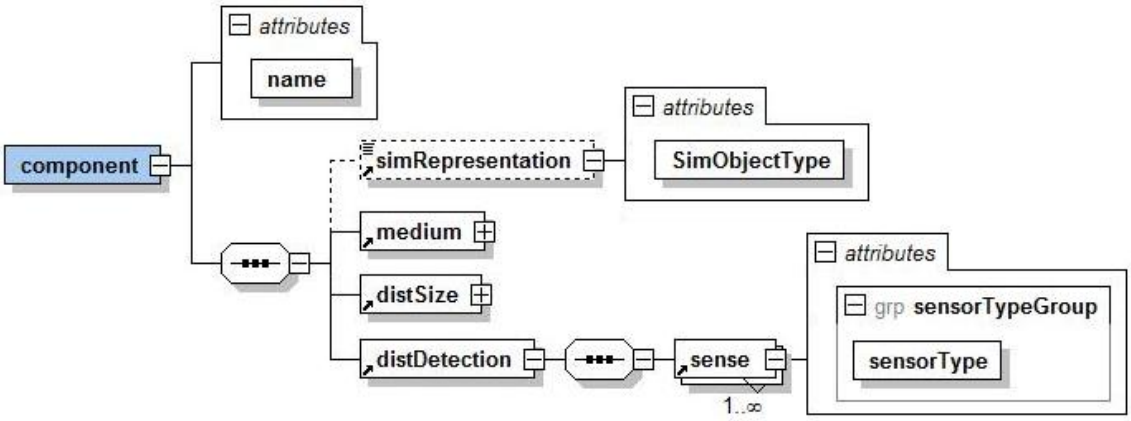


Figure 4.23: Components of a Disturbance

The *simRepresentation* element identifies a possible representation of the component within the visual simulation platform. It is comprised of the type of object that represents it (be it a vehicle, a person or a visual effect), and the title of the object (which uniquely identifies an object or a visual effect within the simulator).

The *medium* element identifies (through its four boolean attributes – land, air, water and underwater) where the presence of the component can be detected, and which type of vehicles can be used in the detection (ground vehicles, aircraft, boats and/or submarines).

The *distSize* element contains a description of how the disturbance's size changes over time. It is comprised of the initial size, optional minimum and maximum sizes, and the growth section, which contains the description of how size changes over time. Alternatively, a predefined dispersion model can be specified for the component – see Fig. 4.24. The initial size defines the size and shape of the disturbance when it first appears. The minimum and maximum size elements, if present, indicate, as the name implies, the limit sizes for the disturbance component. When present, these specifications will be used to determine component size, in addition to growth or dispersion definition. The growth section details size or dispersion model, again using a number of predetermined generic functions and coefficients to customize those functions (with variables such as distance from center of the disturbance being considered int he case of dispersion model). In the case of dispersion, an external dispersion model can be specified as an alternative. There is a variety of dispersion models that can be used, such as AERMOD [Cimorelli et al., 2004], CALPUFF [Scire et al., 2000], BLP [Schulman & Scire, 1980] or OCD [DiCristofaro & Hanna, 1989], among many others. These models are used by governmental and

non-governmental agencies to predict and simulate the dispersion of certain elements in the atmosphere or in the water.

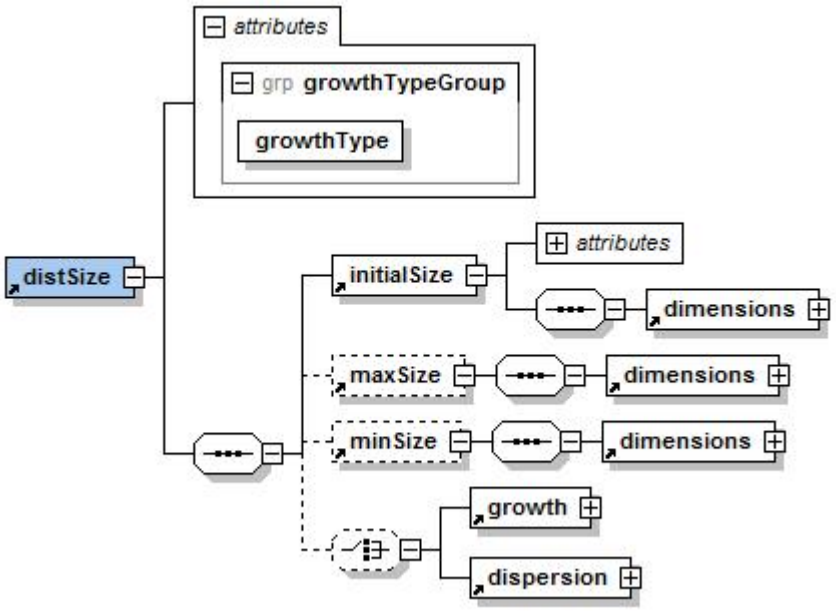


Figure 4.24: Disturbance Component Size

Finally, the list of sensors that can be used in the detection of the specific component of the disturbance contains the identification of sensor type. This sensor type can also be found in the team definition, when listing the sensors each vehicle is equipped with (see section 4.3.2). This information will later be used to match the vehicles that can detect each disturbance.

4.5 Mission Description Language

In this section, a full overview of the Mission Description Language (MDL) is given, and each of its elements detailed. In the context of this work, a mission can be described as a task or set of tasks that should be performed by a team of vehicles. It is comprised of a denomination, a description, and a set of phases, as well as an indication of the team that should perform the mission, as shown in Eq. 4.6 and depicted in Fig. 4.25.

$$Mission = \left\langle team, denomination, description, \{Phase\}^{1+} \right\rangle \tag{4.6}$$

Each phase of the mission has a unique identifier and several other elements (phase type, denomination, description, areas, requirements, tips and targets), as can be seen in Eq. 4.7, and represented graphically in Fig. 4.26.

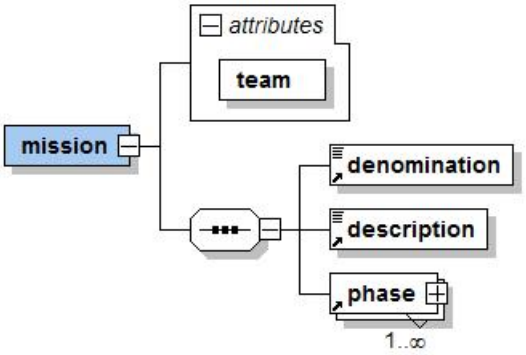


Figure 4.25: Mission Element Definition

$$Phase_{ID} = \left\langle type, denomination, description, \{Area\}, Requirements, Tips, \{Target\}^{1+} \right\rangle \quad (4.7)$$

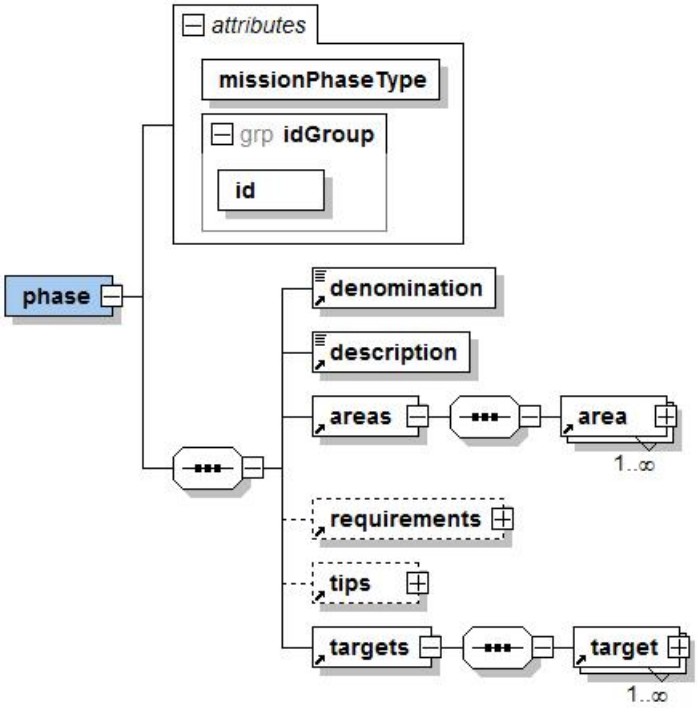


Figure 4.26: Mission Phase Element Definition

The type of phase (*missionPhaseType* attribute) determines the order and priority of phase execution. There are three phase types – base, conditional and extra. Base phases are to be performed when the mission is sent to the team. Conditional phases are to be performed only when certain conditions are met, namely when one or more of the other phases have already been performed (more information regarding conditional phases can be found below). Finally, extra phases are phases with a low priority, that are to be executed either on their own, or after the

execution of some other phase(s); these extra phases, however, are only performed when there are enough resources available, considering that base and conditional phases have higher priority regarding the use of team resources.

Each phase has its own denomination and description, as well as a set of areas of action. These areas specify the geographical areas where the phase will take place. For instance, in the case of a mission with a phase that involves the search for a forest fire, the area would specify the forest area where the search would be performed. Vehicles, however, are still subject to mobility restrictions within this area, if no-fly areas have been specified in either the SDL or TDL files, that intersect the areas of the mission phase. The definition of the *area* element is the same as already presented above (see section 4.2.5).

4.5.1 Phase Requirements

Phase requirements represent hard constraints, that have to be met in order for the phase to be performed. There are three types of requirements, as can be seen in Fig. 4.27:

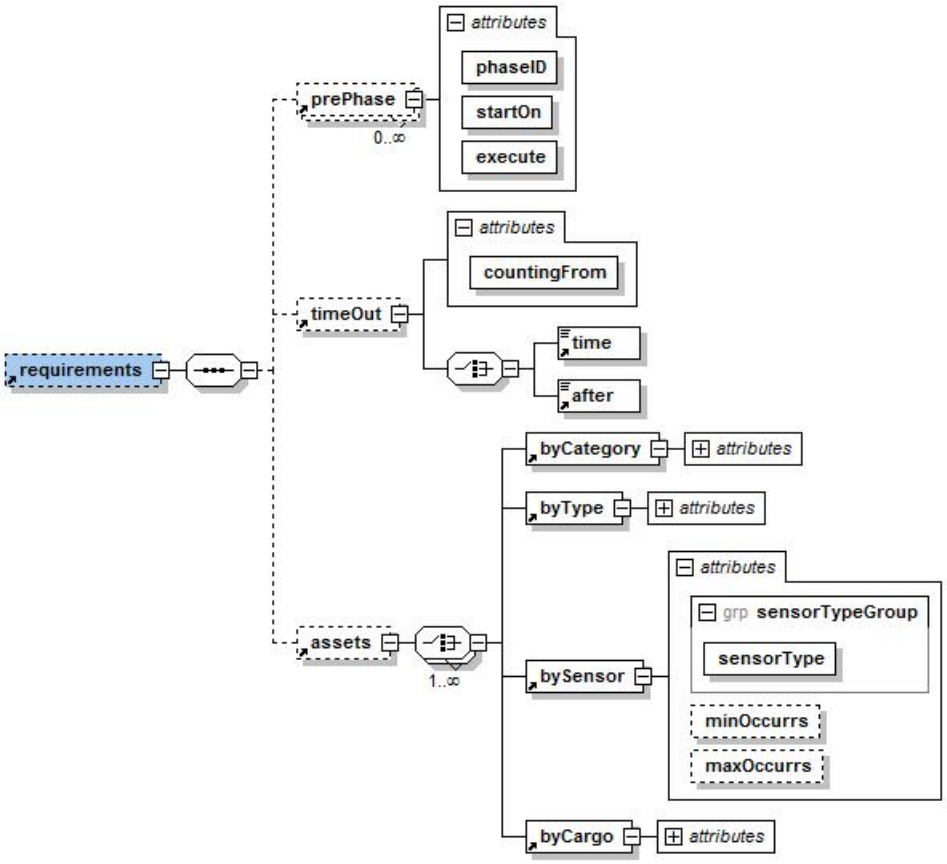


Figure 4.27: Phase Requirements

- **Predecessors**. If present, these requirements specify the phase or phases that have to be performed before the current phase can be executed. The *startOn* attribute can take the

values *One* or *All*, which means that the phase will be executed either when one or all of the targets of the predecessor mission has been accomplished, respectively. The *execute* attribute can take the values *Once* or *ForEach*, which means that the current phase will be executed only once or every time a target of the predecessor phase has been accomplished, respectively.

- **Temporal**. If present, this requirement indicates the maximum amount of time the team has to perform the phase. The *timeout* element, similarly to what was presented in the disturbance availability (section 4.4.2), can be specified using an absolute time or a relative time duration, counting from simulation start.

- **Assets**. If present, these requirements indicate the assets that must be available for the phase to be performed. There are four asset types that can be specified – vehicle category, vehicle type, sensor and cargo. Vehicle category (aircraft, car, boat or submarine) specification includes the category and minimum and maximum values (for instance, between two and four aircraft). Vehicle type (as defined in the SDL file – see section 4.2.7) definition includes the reference to the specific vehicle type as well as minimum and maximum number of vehicles of such type. Sensor definition includes the sensor type (as defined in the TDL file, when specifying the team of vehicles – see section 4.3.2), as well as minimum and maximum number of sensors of the specified type. Finally cargo definition includes the type of cargo (as defined in the TDL file also), as well as minimum and maximum quantities of the cargo type.

4.5.2 Phase Tips

Phase tips represent soft constraints, that should (but do not necessarily have to) be met during phase execution. Phase tips are of five types, as depicted in Fig. 4.28:

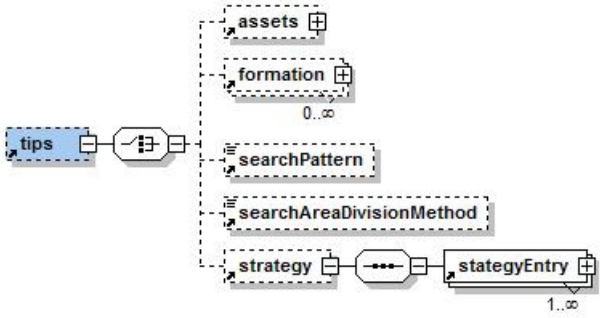


Figure 4.28: Phase Tips Element Specification

- **Assets**. Presenting the same definition as above, for phase requirements, this allows desirable assets to be specified.

- **Formation**. This section of the phase tips allows for the definition of one or more formations for the team. A formation is the physical distribution of vehicles, and it is assumed that motion is to be performed in a synchronized manner, as to maintain the physical arrangement. Formations can be specified using two distinct formats:

  – **Leader and Followers**. This specification method involves the definition of a leader for the formation, which can be either a vehicle, or the mass center of all vehicles. The position of each of the following vehicles is specified as a position relative to one of the previously specified positions, be it the leader or one of the followers, as depicted in Fig. 4.29.

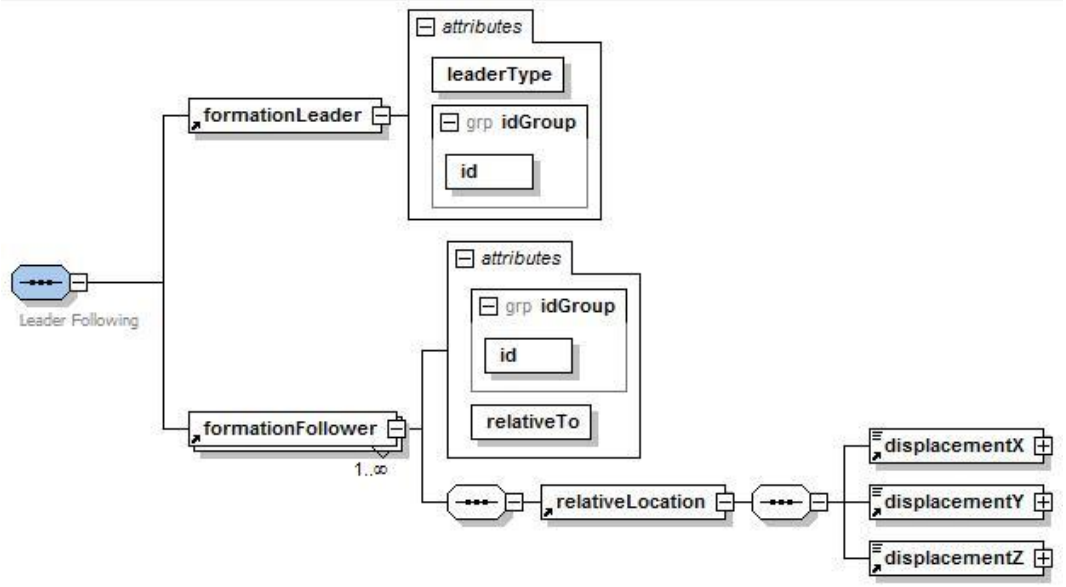


Figure 4.29: Formation Specification with Leader and Followers

  – **Grid**. This specification method involves the definition of occupancy levels for cells over a grid-divided space. First, vertical layers are specified, each layer defined by an altitude/depth level and the altitude variation (the layer spans from $altitude - variation$ to $altitude + variation$). For each vertical layer, a set of frontal layers are specified. Frontal layers are layers perpendicular to the direction of movement, and specified by the frontal deviation (in respect to the front, back or center of the formation grid) and variation. For each strip of space defined by the intersection of the vertical and frontal layers, side layers are specified. Side layers are the division of space in layers parallel to the direction of movement, and specified by the lateral deviation (in respect to the left, right or center of the formation grid) and variation. In each of the specified cells, the occupancy levels are defined, using the *targetDensity* attribute, which defines the percentage of vehicles that should occupy the specific cell. Also, the minimum and maximum number of vehicles present in each cell can be specified. Figure 4.30 shows

the definition of formation using a grid representation in a graphical notation and Fig 4.31 shows a visual example of space grid division.

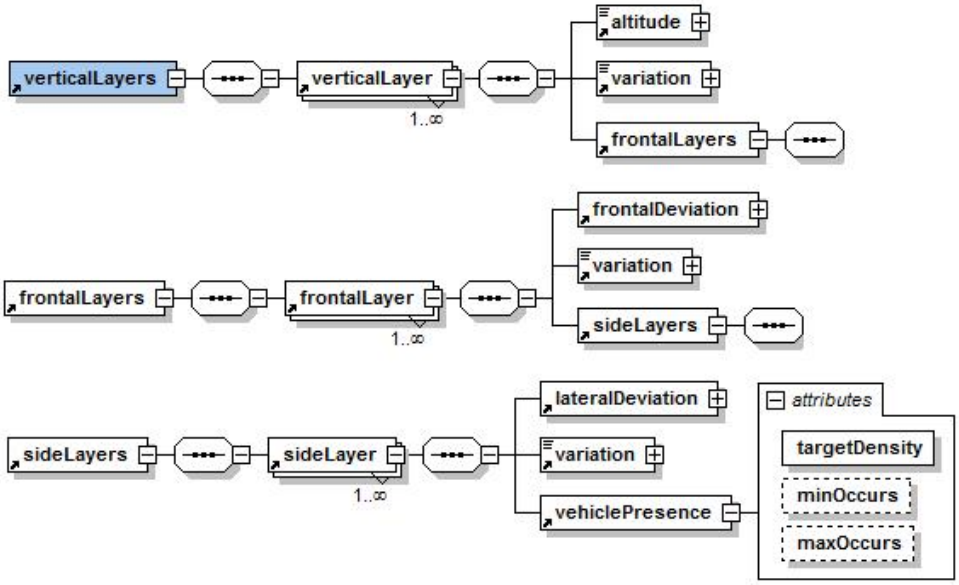


Figure 4.30: Formation Specification with Grid Occupancy Levels

- **Search Pattern**. This element, if present, specifies the preferred method of search (assuming that the phase involves searching for a target). Several predetermined search patterns can be specified, including spiral (inwards or outwards), parallel search, grid search, and others. The best search strategy for a given scenario depends on several factors, such as the size and shape of the search area, the nature of the search object, weather conditions, sensor range and accuracy, among others.

- **Search Area Division Method**. This element, when present, specifies how the search area should be divided among the several vehicles performing the search. Some methods can be specified, including single, quadrant and dynamic. The *single* value specifies that the area should not be divided, and that the vehicle or vehicles performing the search should all operate on the determined areas. The quadrant and dynamic methods specify other division methods, the first with a static division of the search area into smaller areas, and the second providing a dynamic division method, according to the number of vehicles performing the search, their location and capabilities (in terms of speed of search and sensors, for instance).

- **Strategy**. In this context, a strategy is considered to be a set of tactics (each tactic can be defined as a set of actions), each with a set of activation conditions. The activation conditions can be combined, using the conjunction and disjunction operators, as depicted in Fig. 4.33.

  Several triggers can be specified, including when a vehicle enters or leaves a determined area, when it is within a certain distance of the target, when it reaches a certain altitude (in

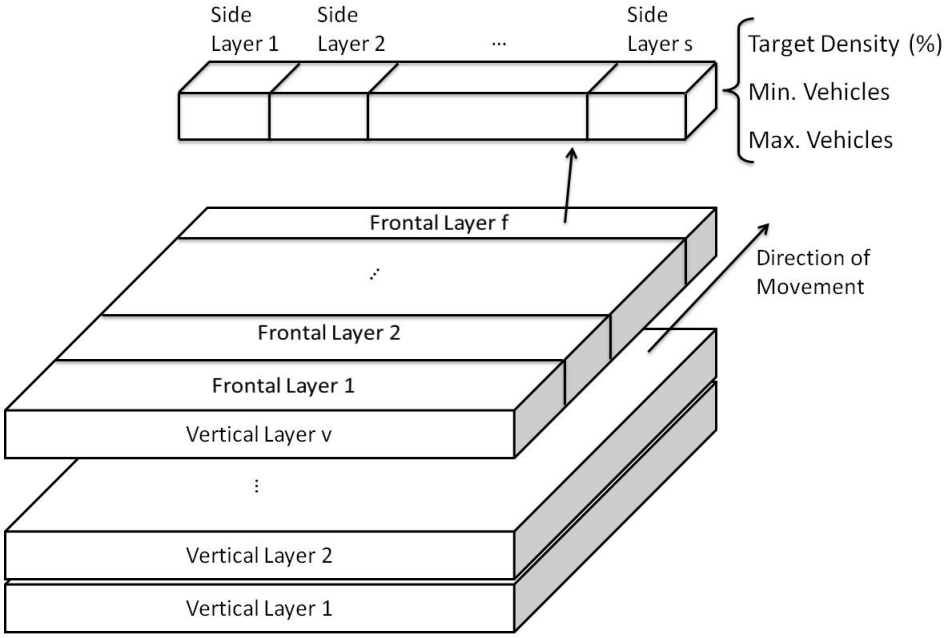


Figure 4.31: Grid Occupancy Levels Diagram

the case of aircraft), or depending on environmental conditions (wind, rain, type of terrain), or even when a goal is achieved (for instance, when a target is found or reached). The several kinds of triggers imply a variety of elements that need to exist to specify its varying parameters, as can be seen in Fig 4.33. For instance, when the trigger is a vehicle entering or leaving an area, the area must be specified. If the trigger depends on weather conditions, such as the speed of the wind, both the wind speed and a comparer (equal, greater or less than) must be specified.

Several actions can also be included in the activated tactic, such as a change in the search pattern, or in the search area division method. A new formation can be specified, using an identifier for a formation, as defined above. Also, an asset can be removed from or added to the set of assets being used in the current phase.

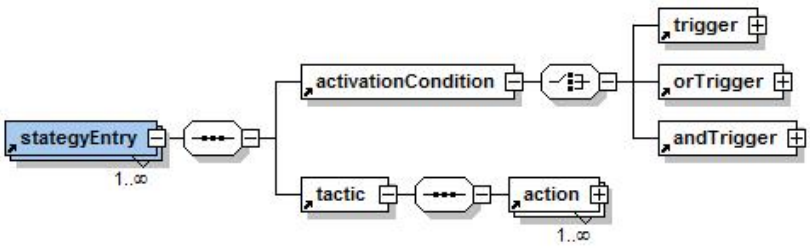


Figure 4.32: Strategy Entry

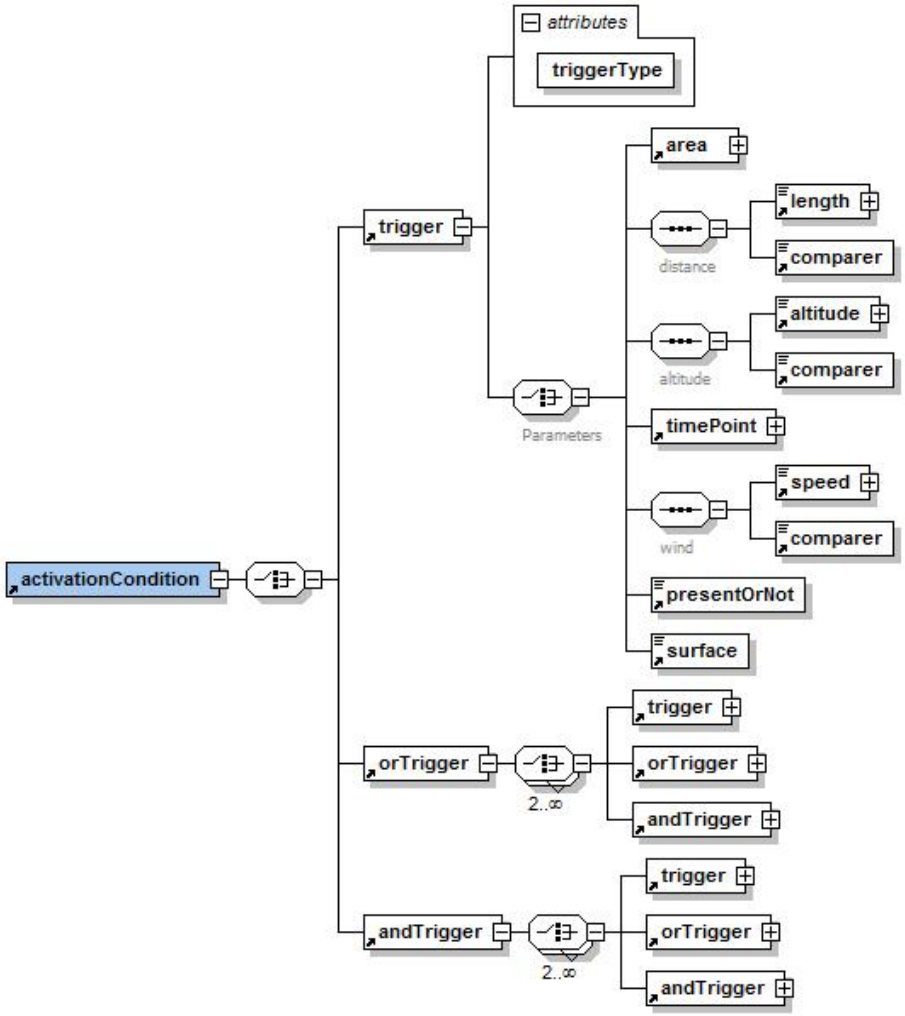


Figure 4.33: Strategy Activation Condition

4.5.3 Phase Targets

Each phase may contain one or more targets, each target specifying an objective. Each target has a unique identifier and several elements, as can be seen in Fig. 4.34

The *from* attribute allows for a target to be specified as an inherited target, from a previous phase. For instance, if the mission consists in detecting and putting out a forest fire, the first phase would consist of detecting the fire, and the second phase would consist on dropping water or fire retardant onto the detected fire. The second phase can only be executed when the first phase succeeds; this is accomplished by defining the phase as a conditional phase and specifying the first phase as a predecessor; assuming that more than one fire may be detected, the second phase would have the values of 'One' and 'ForEach' for the *startOn* and *execute* attributes, respectively (see section 4.5.1); the target for the second phase is only determined during mission execution, and as such it must be inherited from the first phase.

The *disturbanceType* attribute indicates what kind of disturbance is being used as a target (as

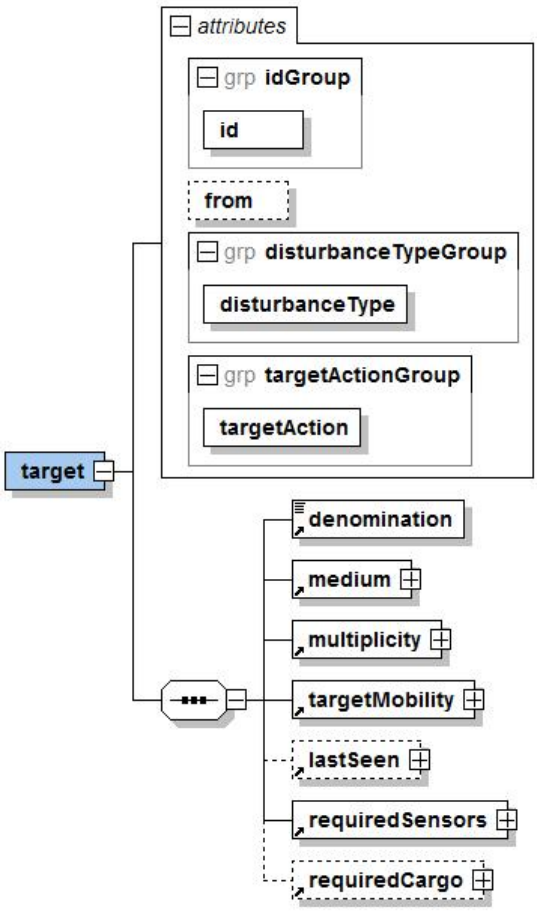


Figure 4.34: Phase Target Definition

specified in section 4.4), and the *targetAction* attribute specifies the type of phase (action to be performed). Examples of actions include Detect (which is used in standard searches, as can be the case of a search for a possible forest fire), Detect Origin (which can be used to detect the source of a pollution cloud, using for instance a gradient search method), Follow (used to follow a mobile target, as can be the case of a vehicle) or Measure (used only to measure values of some sort, as can be the case of chemical concentrations).

The *medium* element, which has the same definition as already presented above, defines where the target can be located, and which type of vehicles can be used to interact with it.

The *multiplicity* element contains two attributes that specify the minimum and maximum occurrences of the target. For instance, and considering the forest fire search example, there is no certainty about the existence of a fire (and therefore, the minimum number of occurrences will be zero), but there is also no limit on the number of fires that may exist (in which case the maximum number of occurrences may be *MAX_INT*).

The *targetMobility* element specifies the mobility type for the target (stationary or mobile), and the growth type (fixed, variable or spread). Both mobility and growth types are just an indication of the most likely behavior of the target (and can, most of the times, be inferred from the

disturbanceType attribute). As such, there is no detailed specification of either mobility or growth (these are only present in the disturbances description file, as described above).

The *lastSeen* element, when present, indicates that the target has been reported in sight, usually by an external agent (when considering multiple missions, it can also be an indication from the previously executed mission). This element contains the time and location of the sight, as well as the confidence level on the observation. Additionally, if the target is mobile, both heading and speed of the target at the time of sight can be specified. Also, if the target has a variable size, the size of the target at the time of the sight can also be specified.

Finally, the *requiredSensors* and *requiredCargo* elements define the necessary sensors to detect the target, and the cargo that should be used in the action (for instance, and returning to the forest fire example, the second phase would require a cargo of water or fire retardant to be specified).

4.6 Conclusions

This chapter detailed the implementation of the four developed languages, for scenario, teams, disturbances and missions description.

These languages have been classified according to their category – static or dynamic. The static category includes the scenario and team definition, while the dynamic category includes the disturbances and mission definition. Furthermore, the orientation of the languages towards scenario or team was also used within each category – both scenario and disturbances definitions are considered to be scenario-oriented, while both team and mission definitions are considered team-oriented.

The scenario description includes a detailed description of the physical scenario in which the mission will take place, as well as several static and control structures. In detail, it includes a description of the available facilities (bases of operations, which may contain an airport, port and/or ground base); the no-fly areas that all teams and vehicles must stay clear of; traffic controllers that may control traffic in a centralized manner in restricted areas; and the vehicle types that exist in the simulation.

The team description contains some generic information regarding the team, the composition of the team (the set of vehicles that compose the team, and their particular characteristics), and the list of facilities the team can use (referencing the bases of operations described in the scenario file). It may also include additional team-specific no-fly areas.

The disturbances description includes a detailed description of each disturbance that exists in the environment, including its initial location and mobility pattern, its temporal availability, and several possible components for the disturbance, each with its own size evolution pattern and detection requirements.

Finally, the mission description contains a list of mission phases, each including the physical areas in which it will take place, a set of possible hard and soft constraints (corresponding to requirements and tips for the mission execution, respectively), as well as a description of the

targets for each phase. The execution of a phase may depend on the successful completion of a previous phase, and the targets may be inherited from those phases.

The four languages were implemented as XML dialects. This fact implicitly endows the languages with several desired characteristics:

- **Readability**. The languages can be easily interpreted by both humans and machines. Since a XML document contains not only the data necessary to describe all intended entities, but also the meta-data associated with it, every piece of data can be easily interpreted. This contrasts with textual representations such as those found in [Peel, 2001] or [Peel, 2009] for the description of airports, that focus on maintaining a low file size, and easing file interpretation by machines.

- **Extensibility**. XML dialects can easily be extended, simply by changing the definition of one or a few elements. By using an automated tool (the XML Schema Definition Tool [Microsoft Corporation, 2010]) to convert the Schema definition into classes usable by the various platform components, these extensions can be mapped into the platform in a relatively fast and easy manner.

- **Data validation**. By using a Schema for each dialect, several validations can be automatically performed by the application when reading the XML file, including structure, data types and values or identifier references, among others. This requires less code to be produced for data validation, thus decreasing both time and effort implementing such validations.

- **System-independence**. Being textual files, XML documents can be easily read by any application in any operating system. Also, more and more programming languages now have some programming interface that provides high-level functionalities for XML documents, including reading and writing operations, validation against a given Schema, mapping to a representation more suited for the language in question, and several editing functionalities.

The current definition of the four dialects includes all entities originally intended to be represented in the languages. Some of these entities were represented with a higher level of abstraction, even though some more detailed descriptions could be included (some examples are listed below).

Even though it seems that all these specifications would take too long to configure for each simulation session (and configuration time being one factor to account for in platforms such as this [Mackenzie & Arkin, 1998]), one has to consider that both SDL and TDL, being classified as static, will rarely change after the initial specification. Thus, both scenario and team configuration are usually only performed once, for the first simulation, and then only when changes are made to either the environment (such as adding a new base of operations, or a new no-fly area) or the team (such as adding a new vehicle, removing an existing one, or editing the bases of operations the team can make use of). Also, a mechanism can be added to the platform so that the state of a team after completing a mission is saved into a new TDL file for future use, thus accounting for a new spacial location of the vehicles, as well as differences in resources (such as fuel or cargo), and

also possible failures. This facilitates team reuse without requiring the operator to specify another TDL file. Also, the DDL and MDL files (or part of them) can be easily reused from mission to mission, considering that most teams will be used for similar missions.

The TDL definition foresees the possibility of specifying more than one team to operate within the defined scenario. This can be used, along with the team attribute in the mission specification, to define different missions for different teams, thus generating a more complex environment (with vehicles from other teams operating on the same environment), and opens the way for the specification of missions that will lead to a competitive behavior among teams.

As mentioned above, some specific extensions have been identified in the language specifications, such as:

- GML could be used to describe areas (no-fly areas and controller areas) in more detail – see section 4.2.5. Even though controller areas and no-fly areas are expressed as either a polygon or a circle, with lower and upper altitude limits (which corresponds to how these areas are usually defined in the real world [FAA, 2010b]), a generalization could be made, as to express other geometrical forms, or even area composition (for instance, applying boolean operations to areas); GML could also be used in the specification of a path (when a disturbance has a motion with a predetermined path);

- The X3D dialect could be used for describing in detail the payload areas and sensor dimensions (or even the rough external aspect of the vehicles, as well as initial, maximum and minimum sizes of disturbances) [Web3D Consortium, 2008]. Even though the information contained in the language definition is enough for the purposes of this simulation platform, a more detailed description of the geometry of payload areas and cargo could help automate the process of matching sensors to payloads, thus avoiding this manual process, and leaving the work to packing heuristics [Allen et al., 2009];

- SensorML [OGC, 2007b] could be used to describe the details of the sensors that each agent is equipped with, thus allowing for higher interoperability [Aloisio et al., 2006]; Its use in the specification of the sensors required to detect a given disturbance component could also help bring the simulation of sensor readings and functionality closer to reality;

- AIXM could have been used in the description of some aeronautical structures, namely within the airport structure definition [Brunk & Porosnicu, 2005];

- MathML could be used in the specification of speed, size and growth / dispersion patterns of disturbances instead of (or in addition to) the use of a set of predetermined function types and corresponding coefficients to express these elements [W3C, 2009]; Even though such specification using MathML would imply a higher complexity, it would also allow for a much more flexible definition of such elements, not being limited to the available functions.

However, a decision was made not to implement these features on the short term (instead using simplified versions of several concepts from these dialects), mainly due to two reasons: on the

one hand, it would represent a level of detail that was not intended in the languages; on the other hand, small tools can be easily developed to make the conversion between dialects (or the language definitions can be changed, as to allow both forms of element definition).

By using the Control Panel (described in section 5.2), it is possible to create and edit files according to these specifications, and send the necessary information to the proper platform components (as described in section 3.2.2). The following chapter describes some of the main platform components, including the Control Panel, with brief explanations on how they make use of the configuration files described in this chapter.

Chapter 5

Platform Main Components

This chapter describes the main components that make up the developed platform – Simulator, Control Panel, Disturbances Manager, Monitoring, Logging and Performance Analysis Tools. The ATC Agent and the Vehicle Control Agent, along with the Agent Communication Platform are discussed in the following chapter.

5.1 Simulator

As mentioned before, the central module of the proposed platform is the visual simulator, which should be as realistic as possible, for the simulation of environment and vehicles. This chapter presents the simulation environment in more detail, starting with the selection process that led to the choice of FSX. Then, some possibilities of the simulator are analyzed. Finally, some operational decisions and adaptations made to the simulator are presented.

5.1.1 Platform Choice

The main focus of the analysis of existing platforms was given to flight simulators. This option was made based on two main reasons – first, since a fluid environment can be considered as the most difficult one to simulate, and given that the simulator should be able to simulate vehicles moving through air, land and water, the focus was given to simulators that can simulate a fluid atmosphere; second, since several of the applications foreseen for the platform under development make use of aerial vehicles (and only a few can be performed using exclusively water vehicles), flight simulators were preferred in the analysis (also, these simulators exist in larger number and are usually easier to adapt to other vehicles).

The requirements for choosing a simulation platform such as the ones under consideration can vary according to the goals of the project. For instance, in [Alexander et al., 2005], the authors analyze the requirements for a simulator to be used in military scenario training. Factors such as

fidelity (how well the simulator emulates the real world), immersion (the degree in which an individual feels absorbed by the experience), presence (the subjective experience of actually existing within the computer-mediated environment) and buy-in (the degree to which a person recognizes the experience or event to be useful for training) are considered. In [Craighead et al., 2007], the authors present a comparison of several simulators, both open-source and proprietary, capable of simulating autonomous vehicles. This study, which includes the simulators presented below, considers factors such as physical fidelity (visual and audio simulation, mainly), functional fidelity (how well the simulator emulates the actual equipment), ease of development (evaluates how easy it is to adapt the simulator to new equipment, existing documentations and the languages that can be used) and cost (both monetary cost and in terms of ease of installation). In [Parodi et al., 2009], the authors analyze the necessary requirements for a simulator in a context of multi-vehicle cooperation, with the possibility of hardware-in-the-loop simulations. The authors evaluate the simulator according to seven categories: multi-vehicle simulation capabilities; inter-vehicle communication model; hardware-in-the-loop simulation; scenario coverage and object creation; environment phenomena; simulation of sensors; and distributed nature and 3D capabilities of the simulator.

In this work, and for a better organization of the analysis, the requirements for the simulation platform were divided into four main categories [Gimenes et al., 2008]:

- **Simulation Engine**. This category was considered the most important one in the analysis, since it pertains to the simulation engine itself. Some less elaborated simulators consider only the four basic vectors that compose a flight: lift, weight, drag and thrust. Drag and lift only exist when there is movement through a fluid, like air. In order to maintain an aircraft in a straight level flight, the lift is equal to weight and thrust is equal to drag [Schiff, 1971]. However, in real flight, an aircraft has to deal with numerous factors, not only pertaining to the aircraft itself, but also external, environmental factors. Aspects such as kinematics, physical simulation, weather simulation and influence on the flight dynamics (weather simulation is especially complex, because some common but not well know factors such as wind shear, turbulence, wind micro bursts, variations of density, pressure and temperature, possibly with severe variations inside clouds, should be taken into account by the weather simulation system), simulation cycle method and others were taken into account when analyzing the simulation platforms.

- **Graphics**. When analyzing game engines, one tends to concentrate on the graphical output of the simulator as the main metric for the analysis. However, in this analysis, this category was considered to be the least important one, since the visual aspect is not the most important factor in scientific simulation. It is, nevertheless, important to analyze the simulators considering this category, given that not only the level of visual detail, realism and attractiveness of the graphics is analyzed but also terrain elevation accuracy and representation, accurate representation of different places and seasons around the world, or even the scene vehicles density the simulator can render smoothly.

- **Fault Injection**. The firm safety requirements related with aviation require the studies of any kind of flight simulation to keep in mind failure considerations. Fault injection is a complex research line to reach a reliable and safe flight model in flight simulation. Basically, the fault injection module is a software module which interrupts the original inputs and outputs of the simulated aircraft. The corrupted inputs our outputs can be generated between the aircraft control (agent) and the flight simulator or the aircraft model and flight simulator (corresponding to failures in aircraft instruments or systems). The quantity of injected faults is a quality parameter of any fault injection method. This category considers the fault injection capabilities of the simulator, the possibility to force equipments, systems and indicators to fail, and the manner in which they do, including failure propagation in dependent systems (for instance, an engine failure should occur some time after a fault in the fuel pumps).

- **Openness**. A simulator is only useful if there is the possibility to interact with it. Hence, software openness is considered in this analysis as a very important category. It takes into consideration features such as the existence of an open API; data import/export protocols; what data is available from the simulator, and what data can be written to the simulator; the possibility to easily develop tools to interact with the simulator, reading and writing data. Existing documentation on possible API and protocols is also considered, as well as the programming languages that can be used.

As previously mentioned, some simulation platforms were analyzed, including some well-known flight simulators, such as X-Plane, FlightGear and Flight Simulator X, briefly described in section 2.2.2. In this section, some additional details are presented.

5.1.1.1 X-Plane

X-Plane uses a geometric approach to determine the flight dynamics: through a process known as blade element theory [Benini, 2004], the aircraft geometry is broken down into a finite number of elements; every simulation cycle, the forces acting on each of these elements are determined and summed for the entire aircraft; using the aircraft's mass and moment of inertia, these forces are converted into accelerations, which in turn allow for the determination of velocities and positions. This method is more accurate than traditional lookup tables, and it allows the simulator to be used in the development of new vehicles, testing their aerodynamics performance. Also, X-Plane includes both subsonic and supersonic flight dynamics (as well as support for flying wings and fly-by-wire systems), and as such virtually any vehicle can be simulated. The weather simulation system is very exhaustive, and it also allows for real-time weather conditions to be downloaded and included in the simulation.

X-Plane's scenery has a world-wide coverage, currently between 60 degrees South and 74 degrees North of latitude, featuring over $33,000$ airports, and totaling over 60GB of scenery information data.

X-Plane has a detailed failure-modeling, with many systems (including instruments, engines, flight controls, control cables, antennae and landing gear) that can be set to fail, either manually or randomly.

X-Plane presents two forms of programmatic interaction: UDP sockets and plugins. Whilst the first method has sparse documentation, allowing for both data retrieval and publication, the second one is well documented, presenting a much wider range of interaction possibilities. Plugins are programs written in C or another binary-compatible language.

5.1.1.2 FlightGear

FlightGear includes three Flight Dynamics Models (FDM), also allowing for the addition of new models or even to interface with external ones. For instance, the Piccolo autopilot system, mentioned in section 2.2.3, has a very realistic simulation engine, simulating many of the forces involved in a flight, but does not present a graphical interface of its own – instead, data from Piccolo can be used directly as an input to a flight simulator such as FlightGear [Vaglienti et al., 2007]. The three default available models are JSBSim[1] (an open source FDM written in C++ using XML configuration files to model aircraft mass, aerodynamic and flight control properties; it models the aerodynamic forces and moments by the classic coefficient buildup method [Berndt et al., 2008]), YASim (this FDM simulates the effect of the airflow on the different parts of an aircraft, in a similar approach to X-Plane; it is possible to perform the simulation based on geometry and mass information, combined with more commonly available performance numbers for an aircraft, which allows for quickly constructing a plausibly behaving aircraft that matches published performance numbers without requiring all the traditional aerodynamic test data) and UIUC (developed by the Applied Aerodynamics Group, Department of Aerospace Engineering, University of Illinois at Urbana-Champaign[2], this FDM is based on NASA's LaRCsim [Jackson, 1995], extending the code by allowing aircraft configuration files and by adding code for simulation of aircraft under icing conditions; it uses lookup tables to retrieve the component aerodynamic force and moment coefficients for an aircraft, using them to calculate the sum of forces and moments acting on the aircraft). The weather simulation system allows for real-world weather conditions to be downloaded and used, but the system works at a global level – local weather conditions are used in the entire world, and thus no local changes are available.

FlightGear features over 12GB of terrain information data, with over 20,000 real world airports represented.

FlightGear does not have a generic failure-modeling system. Many aircraft, however, have some failures modeled into them, and many instruments support failure modeling (although in various ways, from a technical point of view).

FlightGear provides a flexible interface, with several protocols that can be used to access internal variables, including serial port, file, or socket communication. The variables that are accessed can also be configured, through an XML file, thus making interaction with this simulator

[1] More information available at `http://www.jsbsim.com/`

[2] More information available at `http://www.ae.uiuc.edu/m-selig/`

very flexible and adaptable to specific user needs. There is, however, one major disadvantage in FlightGear, which is documentation – it is often insufficient and is scattered over the web, making it difficult for a developer to choose the appropriate protocol, and to learn its peculiarities.

5.1.1.3 Flight Simulator X

FSX uses a parametric approach (also known as behavioral simulation) for the flight model, using a set of parameters, which are completely independent of the visual model [Goodrick, 2000]. The weather simulation system is very comprehensive, also allowing for real-time weather conditions to be downloaded from the internet.

FSX features approximately 11GB of terrain information (including 38 high-detail cities), with more than 24,000 airports (including 45 high-detail airports). This version of Flight Simulator also includes the 'dynamic living world' feature, which includes several airport services (with a diverse range of vehicles, jetways and other features), auto traffic on highways, sea traffic (boats and ships, including aircraft carriers) on lakes and oceans, AI air traffic, and simple herds of livestock or wild animals, all contributing, along with the use of DirectX 10 technology, to an increased level of realism.

FSX, like X-Plane, also supports failure-modeling, featuring over 50 systems that can be set to fail, either manually or randomly.

FSX's Deluxe version includes an SDK, featuring among other tools, the SimConnect API [Microsoft Corporation, 2008b]. SimConnect provides a flexible, powerful and robust client-server communications protocol that allows asynchronous access to hundreds of simulation variables and events. Some of these variables can only be read, but nearly two hundred can also be written to, and several hundred events can be sent to the simulator. There are also several dozens of functions to deal with the weather system, missions, in-game menus, AI objects, communication and data access and manipulation, among other useful features. It also includes extensive and comprehensive documentation, including several functional examples[3]. One advantage of SimConnect is that it allows the developer to choose the implementation language from a large set of possibilities, given that not only C/C++ is supported, but also any .Net-aware language. Also, an unofficial adaptation of the API to the Java language, called jSimConnect, is being developed, thus expanding even further language compatibilities[4].

One of the peculiarities of FSX is the structured experiences system (or mission system), which allows regular users to have a different, interactive experience of flight, with measurable goals, other than to simply fly around from one airport to the next. FSX includes numerous missions, ranging from tutorials that teach the user how to fly an aircraft, to racing missions, simulating the Red Bull Air Race environment, as well as several transport or rescue operations, among others. The mission system allows for the definition of objects, areas, triggers and actions that can be linked to work together in an orchestrated manner, producing realistic, diverse and

[3] More information available from Microsoft Developer Network (MSDN), at `http://msdn.microsoft.com/en-us/library/cc526983.aspx0`

[4] More information available at `http://lc0277.nerim.net/jsimconnect/`

complex missions. Microsoft has also already recognized the advantages of simulation in various business areas and the potential of these structured experiences, and is commercializing the engine behind FSX as an enterprise-oriented product, called ESP[5].

5.1.1.4 Comparison and Choice

In what regards to the simulation engine, FlightGear is the most flexible one – it is equipped with three primary flight dynamics models and it supports the use of an external model, acting only as a visualization tool. X-Plane uses the geometric approach and also supports the use of an external FDM. FSX's FDM uses the parametric approach (lookup tables). Although a geometric approach to the FDM is sometimes preferred, the underlying quality of an engine with a different approach can prove to be as close or even closer to reality – in a simple comparison between FSX and X-Plane by Stock [Stock, 2007], FSX's engine emerges as more realistic. All three simulators are capable of recreating real-time weather conditions, by connecting to different servers around the globe that provide with the necessary data. X-Plane and FSX, however, seem to present more realistic and comprehensive weather simulation systems, when compared to FlightGear.

Concerning the graphical aspects, FlightGear, X-Plane and FSX all achieved a good score, each with its peculiarities (X-Plane, for instance, presenting the most detailed scenario, with an impressive dataset of terrain information for the entire world). FSX seems to overcome the lack of detail (parts of the scenery lack the accuracy of both terrain elevation model and textures) with a decent scenery auto-generation engine, and providing the airports with many vehicles that mimic real-world activity in an airport. There are many available scenery extensions, either commercially or free of charge, that provide a specific simulator with more accurate terrain elevation meshes, more detailed textures, sometimes even including buildings, bridges, and other structure. There are also other kinds of extensions, such as detailed models for airports, new vehicle models, AI traffic packages, sound effects, and even missions, in the case of FSX. These extensions exist in a wider variety and number for FSX than other simulators, most likely due to the long history of success of this simulator and the large existing fan base.

In relation to the fault injection aspects, X-Plane provides equipment failure for many systems (over thirty-five in the previously analyzed version, but more in the current release), and FSX allows failures in over fifty systems, sensors, instruments or other equipment. However, in both these simulators, there is no possibility to vary the manner in which these systems will manifest the failure.

On the subject of openness, all flight simulators offer expansion possibilities. FSX stands out in this category, with the SimConnect API, its extensive documentation, functional examples, and the possibility to choose the implementation language from a large set of possibilities.

The comparison of the three simulators is presented in Table 5.1

With the three simulators achieving similar global evaluation, the choice fell on FSX. This choice also took into consideration the mission system featured in FSX, given the possibilities it

[5]More information available at `http://www.microsoft.com/esp/`

		X-Plane	FlightGear	FSX
Engine	FDM	Geometric + External	3 Native + External	Parametric
	Weather	Good	Local	Good
Graphics	Terrain	60GB	12GB	11GB
	Airports	33.000	20.000	24.000
Fault Injection		Yes, several	No, only aircraft specific	Yes, over 50
Openness	Method	Yes, UDP + Plugins	Yes, very Flexible	Yes, SimConnect
	Doc.	Good	Poor	Excelent
Extras		FAA-Certified	Open-source	Mission System

Table 5.1: Simulator Comparison Summary

provided for this particular platform; the fact that the SimConnect environment kit allows not only the manipulation of terrain, traffic or weather, but also of special effects, such as fire, explosions, smoke and many others; and the large community of users developing new vehicles, and other add-ons for this flight simulator.

The chosen environment can be categorized as partially observable (each aircraft can only sense the environment surrounding it, and the sensors are subject to noise or malfunctions), stochastic (the behavior of the aircraft is defined in the respective lookup tables, which could be considered deterministic, but there are many external variables, which are not pre-defined, that influence the final result), sequential, dynamic, continuous (the time component can be considered discrete, since the simulation is based on cycles) and multi-agent, according to the categorization described in section 2.1.2.

5.1.2 Simulator Possibilities and Applications

As previously mentioned, Microsoft Flight Simulator X has been used by researchers in several projects, in diverse areas such as fluid dynamics [Kenny et al., 2008] or geographic databases [Diehl et al., 2009], along with more traditional areas, more closely related to flight simulation [de Farias et al., 2007] [Cantoni & Neto, 2008].

5.1.2.1 Simulation Limitations and Possible Solutions

One of the limitations of the simulator is that it only simulates a circular area with 200km of radius, centered on the user aircraft (the user aircraft is the aircraft that the player controls). This limitation, however, can be overcome by using more than one instance of the simulator and its multi-player capabilities – by strategically placing user aircraft in different airfields on the different instances of the simulator, the covered simulation area also increases. Figure 5.1 shows an example of how two instances of FSX can be used to cover the entire continental area of Portugal – one user aircraft is placed on the Viseu airfield (ICAO code LPVZ) and the other on the Beja Airbase (ICAO code LPBJ).

The platform vehicles have to be created in the simulator that covers the vehicle's initial location, and a handling protocol has to be created, as to transfer the vehicle from one simulator

Figure 5.1: Continental Portugal Area Coverage with Two Instances of FSX

instance to the other, when the vehicle moves outside the simulation radius of the first simulator. This situation, however, introduces difficulties in the operation of ATC Agents – if an ATC Agent is responsible for an area that is not fully covered by one simulator, it has to connect to more than one simulator, and manage the simultaneous connections and all the data received from the simulators. Similarly, most of the remaining components also need to be adapted to support this feature.

5.1.2.2 Emotional Feedback and Control

One of the applications of the simulator was shown in [Silva et al., 2009b]. The authors demonstrated that the simulator can be used, together with an emotional assessment tool (described in [Vinhas, 2010]), to trigger and reflect emotional responses. Taking advantage of recent developments in fields such as sensor miniaturization, wireless communications and immersive simulation environments, the authors envisioned an integrated multimedia fully bidirectional interactive system where system's parameters were directly changed accordingly to user's emotional response. The real-time automatic emotion assessment achieved with low intrusion levels is of great importance for social robotics as it would surely potentiate both physical and ubiquitous interaction. Such system capability would provide the ability of generating intelligent environments that would fit the user's emotional states [Kim et al., 2009].

By using this multimedia system, the authors were able to provide distinct practical scenarios to apply in several situations that range from traditional entertainment applications, through immersive realistic animations contextualized with user's emotions, to therapeutic phobia treatment

– according to a poll by CNN and Gallup for the USA Today in March 2006, 27% of U.S. adults would be at least somewhat fearful of getting on an airplane [Stoller, 2006]. Several solutions are offered to treat this phobia, including medication, and some behavior therapies, including virtual reality solutions. These solutions are often used in conjunction with a more conventional form of therapy [Kahan et al., 2000] [da Costa et al., 2008]. One such example is Virtually Better, a clinic which offers several solutions based on virtual reality technology to support therapy in anxiety disorders [Rothbaum et al., 2006]. However, and despite having around fifty clinics worldwide – the majority located within the United States – it cannot offer its solutions to a very wide audience at an affordable cost. Some companies, such as Virtual Aviation, offer an even more realistic experience, using the same multi-million dollar simulators used to train professional pilots [Bird, 2005]. Such companies have an even more limited geographical availability, and prohibitive prices – up to three thousand dollars for a session.

One of the major models of emotion representation is the Circumplex Model of Affect proposed by Russell. This is a spatial model based on dimensions of affect that are interrelated in a very methodical fashion [Russell, 1980]. Affective concepts fall in a circle in the following order: pleasure, excitement, arousal, distress, displeasure, depression, sleepiness, and relaxation – see Fig. 5.2. According to this model, there are two components of affect that exist: the first is pleasure-displeasure, the horizontal dimension of the model, and the second is arousal-sleep, the vertical dimension of the model. Therefore, it seems that any affect stimuli can be defined in terms of its valence and arousal components. The remaining variables mentioned above do not act as dimensions, but rather help to define the quadrants of the affective space. Despite the existence of criticism concerning the impact of different cultures in emotion expression and induction, as discussed by Altarriba [Altarriba et al., 2003], Russell's model is relative immune to this issue if the stimuli are correctly defined in a rather universal form. Having this in mind, the Circumplex Model of Affect was the emotion representation abstraction used in the proposed project.

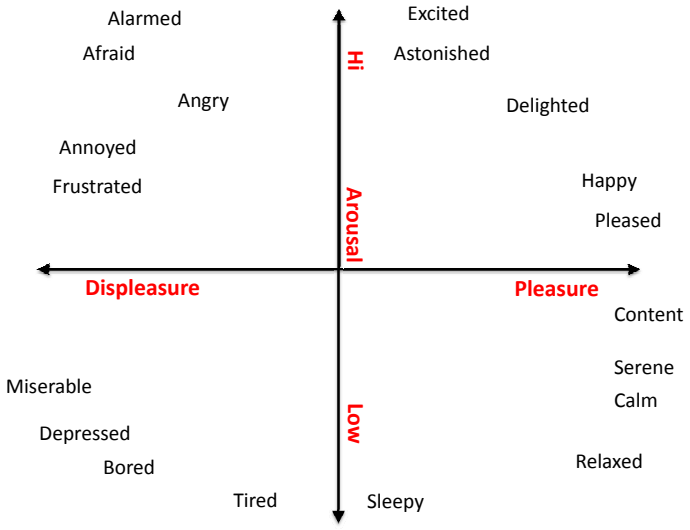
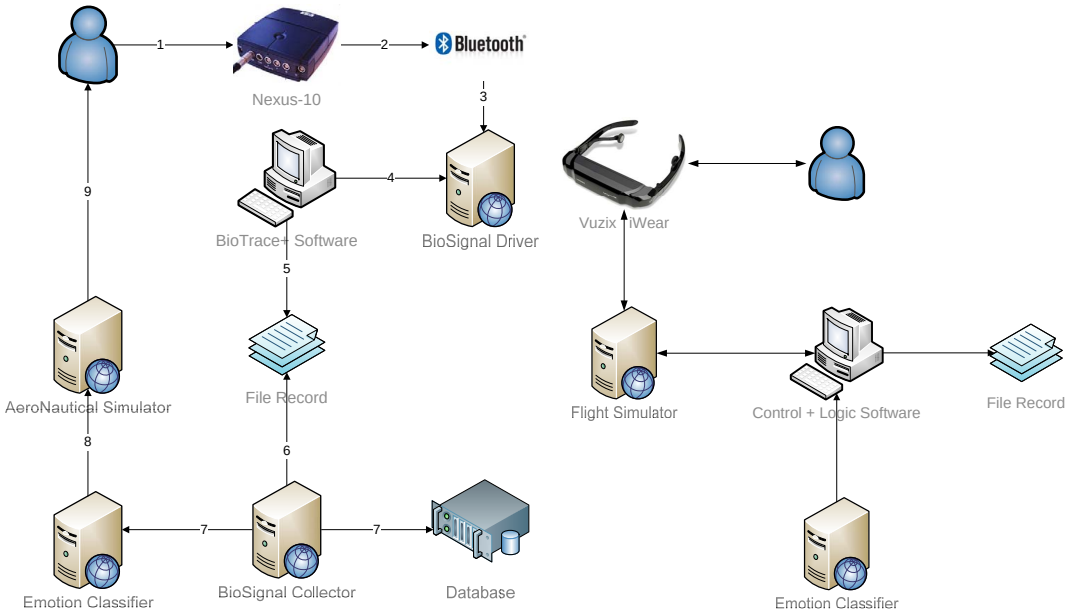


Figure 5.2: Russell's Circumplex Model of Affect [Russell, 1980]

Architecture and Module Description The system global architecture is based on independent and distributed modules, both in logic and physical terms [Silva et al., 2009b]. As depicted in Fig. 5.3(a), and following its enclosed numeration, it is possible to appreciate that biometric data is gathered directly from the subject by using Nexus-10 hardware. In more detail, temperature, GSR and respiration sensors are used – from these sensors, phalanx skin temperature, direct galvanic skin response and respiration amplitude and frequency rates are computed. In order to reduce the number of wires presented to the user, and therefore reduce the impact of signal measurement, thus conserving immersion sensation, the collected biometric data is, in real-time, transmitted by Bluetooth to a computer in the proximities running the adequate data driver. The next step is of the responsibility of BioTrace+ software, supplied with Nexus-10, and beyond providing a configurable interface for online signal monitoring, it records biometric data directly as an accessible text file.



(a) System's Global Architecture (b) Aeronautical Simulator Module

Figure 5.3: System Global Architecture (a) and Aeronautical Simulator Module (b)

The denominated BioSignal Collector software was developed in order to access the recorded data in real-time and make it fully available for further processing either by database access or online TCP/IP socket connection. In this last category, lies the Emotion Classifier, as it is responsible for online user's emotion state assessment – how this process is conducted is fully described in the next subsection. The continuous extracted emotional states are projected into the Russell's model quadrants and are filled as inputs for the Aeronautical Simulator. This system module, as a cycle of its own, as described in Fig. 5.3(b), and briefly depicted in the next paragraph.

The simulation endpoint, which serves as a running example, has a simple architecture. The main module communicates with the emotional endpoint and receives data from the emotion assessment module, indicating which of the four quadrants of the Russel's Model should be active.

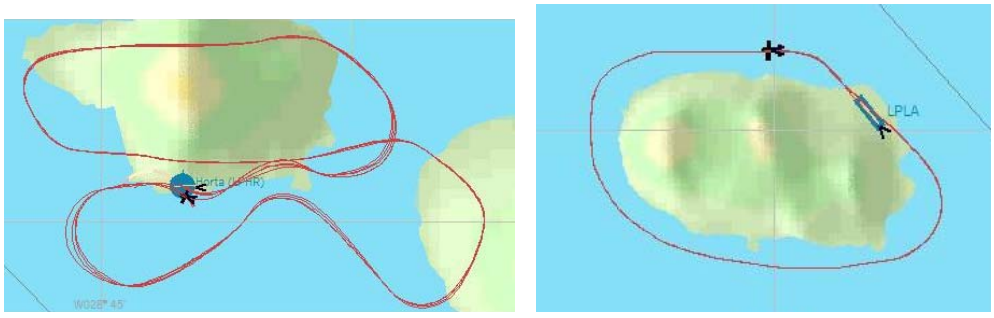
The module, in turn, communicates with the chosen simulator, changing its internal variables in order to match the desired quadrant, and as explained in more detail in section 3.3. This module also produces a permanent accessible log file, with information collected from the simulator regarding location and attitude of the user plane. The simulator interacts with the user through immersive 3D video hardware, which allows the user to control the visualization of the simulation.

Experimental Settings The desired emotional quadrant influences the simulation in three dimensions: weather, scenery and maneuvering [Silva et al., 2009b].

The two quadrants characterized by a state of displeasure are associated with worse climacteric conditions, ranging from heavy thunderstorms, in the one related with fear (quadrant 2), to foggy cold fronts, in the one related with boredom (quadrant 3), leading to a rougher flight. The two quadrants characterized by the feeling of pleasure are associated with fair weather, producing a more stable flight.

The chosen global scenery is an archipelago, more specifically, the Azores archipelago, a set that can provide both a pleasant flight, with many enjoyable sightseeing moments, and an irregular one, crossing a major thunderstorm trying to keep the plane leveled.

For the two quadrants associated with high levels of arousal resultant of either excitement or fear (quadrants 1 and 2 respectively), the chosen itinerary takes the plane around an island, with many closed turns, at low altitudes, ranging from five to twenty-five hundred feet, including a low-altitude pass over the major city in the island. This roughly eight-shaped path also includes a brief incursion into a second island in close proximity to the first one, as shown in Fig. 5.4(a).



(a) Route for Quadrants 1 & 2 (b) Route for Quadrants 3 & 4

Figure 5.4: Routes for Quadrants 1 & 2 (a) and 3 & 4 (b)

For the two quadrants associated with low levels of arousal, the chosen itinerary consists of an oval-shaped route around an island, as shown in Fig. 5.4(b). The turns in this route have a superior radius (resulting in smaller aircraft roll angle) and the altitude variations have smaller amplitude. As a result, the flight is experienced as a calmer one. Closely related to the route description is the maneuvering control. All maneuvers are done via the autopilot system present in the simulated aircraft. Given the waypoint the plane must follow, the desired heading is calculated, using the Great Circle formulas, and adjusting the heading to the magnetic declination of the area in which the flight takes place, this value is used as the input to the heading control of the autopilot system.

For the first route, typical autopilot controls are active, namely speed, heading and altitude, which controls the speed of the aircraft, the direction in which it should be flying and the altitude, respectively. As for the second route, two extra features are applied – maximum bank and yaw damper. The first limits the maximum roll angle of the plane during turns, while the second reduces rolling and yawing oscillations, making the flight smoother and calmer.

Table 5.2 shows a summary table of how the simulation environment is influenced in each of the three dimensions for each quadrant. A mapping between quadrant numbers and the respective quadrants in Russell's model (see Fig. 5.2) is included.

<table>
<tr><th>Quadrant</th><th>Russell's Model Quadrant</th><th>Scenery</th><th>Weather</th><th>Maneuvering</th></tr>
<tr><td>1</td><td>Upper Right</td><td rowspan="2">Eight-shaped (Fig. 5.4(a))</td><td>Fair Weather</td><td rowspan="2">Typical AP</td></tr>
<tr><td>2</td><td>Upper Left</td><td>Heavy Thunderstorms</td></tr>
<tr><td>3</td><td>Bottom Left</td><td rowspan="2">Oval-shaped (Fig. 5.4(b))</td><td>Cold Fronts</td><td rowspan="2">Typical AP + Max Bank + Yaw Damper</td></tr>
<tr><td>4</td><td>Bottom Right</td><td>Fair Weather</td></tr>
</table>

Table 5.2: Simulation Environment Influence Summary

Sensors for skin temperature, galvanic skin response and respiration rate and amplitude were used. In order to present the user with an immersive experience, 3D video hardware was used, in the form of virtual reality video eyeware. This equipment provides the user with a three degree of freedom head-tracker, allowing the user to experience the environment as if he was actually there.

After providing the authors with some background information to characterize the sample, the user was then connected to the biometric equipment, and an emotional baseline was established. Depending on whether the user suffered from pterygophobia or not, an emotional policy was followed by the operator. This policy, as already mentioned, ensured that users who suffered from this phobia would not be placed in a situation of high emotional stress, which could exacerbate the fear of flying, but would allow people who enjoy flying to experience extreme situations that could trigger an emotional response.

The experiments were comprised of three distinct sequential stages, as with actual flights. In the first phase, the plane takes off from an airport in one of the islands. The choice of the airport to takeoff from was primarily based on whether the subject stated to suffer from fear of flying. For individuals suffering from pterygophobia, the operator handling the emotional assessment module forced either the third or fourth quadrants, providing the subject with a calm takeoff and flight, as not to trigger an anxiety attack. For the remaining individuals, the operator forced one of the first or second quadrants, trying to obtain an increased amplitude of emotional responses. After takeoff, a series of closed circuits was performed, as already explained above. Finally, in the landing phase, the plane lines up with the selected airport, makes the approach and lands.

The experiments were conducted among thirty-seven subjects, twenty-four male and thirteen female, between the ages of twenty and fifty-six. Seven of the subjects stated that they had some level of fear of flying, while the remaining thirty declared not to be afraid of flying. Of the seven subjects suffering from some form of pterygophobia, four of them revealed that they have in fact

never flown, with only three actually having suffered from the symptoms usually associated with this phobia.

After concluding the trial, the subjects were then asked to describe the experience, the emotional response that the simulation triggered, and if there were occasions when those reactions were stronger.

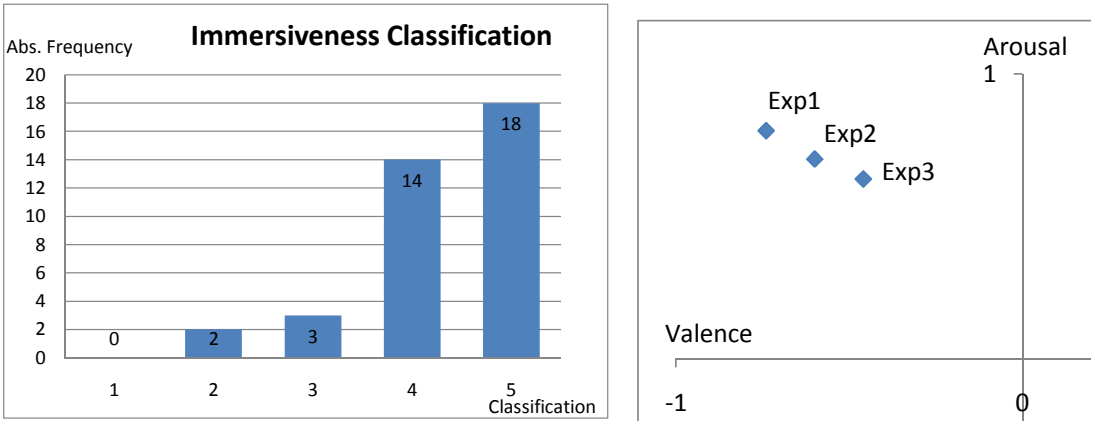
For the case of the seven subjects that stated to suffer from fear of flying, they were asked to repeat the experiment, as to obtain results that could enlighten the authors as to the possible usage of this tool in phobia treatment.

Results and Conclusions The results can be analyzed in three perspectives – emotional assessment, simulation immersiveness and mitigation of the fear of flying

As for emotional assessment, a classification based on self-assessment resulted in a success rate of 77% when considering eight regions in Russell's model, and 86% considering only the four main quadrant[6]

As for the simulation, users were asked to describe their experience, and to classify, in a scale of one to five, the level of immersiveness of the simulation environment. The results shown in Fig. 5.5(a) show that the majority of the individuals considered the environment to be highly immersive, with an average classification of 4.3 out of five.

These results are supported by the subjects' emotional response to the change of the desired simulated quadrant by the emotional assessment module operator, confirming that the simulation environment triggers emotional responses.



(a) Simulation Immersiveness Classification

(b) Emotional Trend

Figure 5.5: Simulation Immersiveness Classification and Emotional Trend

In what regards the aeronautical simulation, it is fair to state that all projected goals where completely fulfilled as users confirmed their immersion sensation either by self-awareness or biological recorded response. It is believed that the usage of 3D glasses as display device played

[6]the detailed results can be found on [Silva et al., 2009b]

a particularly important role in creating the appropriate environment. Also the defined scenarios, with distinct weather conditions, geographical context and maneuvers smoothness, lived up to challenge, as they generally triggered the desired emotional responses. Considering the economical cost of both the simulator and the virtual reality video eyeware used to provide the user with this immersive simulation environment, as well as the high immersiveness level reported by the test subjects, it is also reasonable to conclude that more economically viable simulators can be used in some less demanding environments, such as in the treatment of phobias.

Regarding the subjects suffering from fear of flying, and who repeated the experiment, some interesting results were obtained. One particular outcome that seems to support the fact that this kind of simulation can be used in the treatment is depicted in Fig. 5.5(b), which shows the average emotional response for each of the three conducted experiments (the image shows the upper left part of Russell's Circumplex Model of Affect). As can be seen, the second and third experiments show an emotional response that tends to move away from the extreme end of the second quadrant, denoting a reduction in the levels of fear registered in the subjects during the latter experiments.

As previously mentioned, the results seem to suggest that a significant mitigation of the symptoms of pterygophobia was achieved among the subjects that referred at least some level of fear of flying. However, additional trials would have to be conducted, with a larger sample, in order to fully backup this conclusion. In spite of this, fear of flying is due from a variety of more specific fears (such as fear of heights, fear of confined spaces, fear of speed and others), and as such, more focused tests should be devised, targeted at each of those particular phobias, for the sake of analysis accuracy.

5.1.3 Simulator Adaptations

FSX was originally designed to be used primarily with aircraft (planes and helicopters). Because of that, some adaptations and additional developments had to be made in order to use land, water and underwater vehicles in the same manner as aircraft [Santos, 2010]. This version of the simulator already has the capability to simulate land and water vehicles. In fact, it includes several vehicle models for cars and other ground vehicles, as well as boats and ships, that are used for the generated land and sea traffic referred in section 2.2.2. However, the SimConnect API refers mainly to aircraft and helicopter systems, and therefore some adaptations had to be performed.

5.1.3.1 Vehicle Types

The first adaptation refers to the use of different vehicle types, and their idiosynchratic characteristics. Vehicle types are configured in the Control Panel (see section 5.2.2), using SDL for their description – see section 4.2.7. The simulated vehicle that represents each of the vehicle types is based on an existing one, with the necessary changes made to its configuration files.

Vehicles are stored in the simulator installation folder, and there is a different folder for each vehicle – see Fig. 5.6. Each vehicle folder contains the configuration files for the aircraft, and a set of folders that contain the information regarding the physical model of the aircraft, the visual

textures that can be used with it, the specific sound effects for the vehicle and information regarding the instruments panels. This modularity helped in the adaptation process, so that new vehicles could be defined based on existing ones.

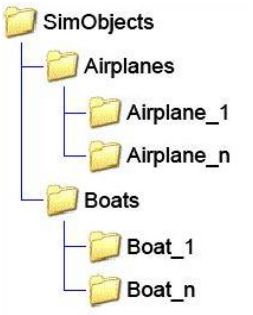


Figure 5.6: Vehicle Folder Organization in the Simulator Installation Folder

When the scenario configuration is launched (step 2 of Fig. 3.10), the vehicle types section of the SDL file is processed and non-existing vehicle types are created by copying the existing vehicle it is based on to a new folder. Then, changes are made to the contents of the vehicle configuration files as necessary, to match the definitions on the SDL file. First, the *simTitle* property, that uniquely identifies a vehicle within the simulator, needs to be changed into a new value; this new value is also replaced in the scenario file (in the simulated agent section of the agent type definition, as seen on section 4.2.7), so that subsequent simulations can use the created vehicle type (and thus avoiding the need to run this process in every simulation). Then, some properties that can only be adjusted in the configuration files are changed. Such properties include the dimensions of the vehicle (length, width and height), its weight when empty or the maximum speed it can sustain (properties that can be found, in most cases, in the physical and performance categories of vehicle type description – see section 4.2.7). Some other properties, available for modification via the programmatic interface, are only changed after the vehicle is created wihin the simulator. Such properties include the amount of fuel and cargo the vehicle currently holds, the state of its lights, and some other properties that can, in most cases, be found in the state description of team vehicles – see section 4.3.1.

Some interpretation considerations had to be made in some cases, as to account for the different vehicle types. As an example, in section 4.3.1 one of the state variables for all vehicles indicates which lights are turned on; when the vehicle in question is a car, boat or submarine, the lights do not have a match to existing variables in the simulator. As such, existing aircraft lights that are not used in other vehicle types are interpreted according to vehicle type, as defined by a vehicle lights translation table.

Some adaptations to the Vehicle Control Agent also exist, depending on the vehicle type. For instance, the helix maneuver (see section 6.3.1.4) is disabled for ground and water vehicles, and thus only available for aircraft, helicopters and submarines; also, the altitude component for the other maneuvers is ignored (or always assumed to be 0 meters above ground level), since the vehicles cannot fly or dive.

5.1.3.2 Submarine Navigation

A special adaptation is required when considering submarine vehicle types. Flight Simulator X was not designed to support underwater navigation and operations (even though it supports water vehicles and amphibious aircraft operations). As such, submarine navigation simulation must take place outside the simulator, between FSX and the Vehicle Control Agent. A small module was built into the Vehicle Control Agent (see section 6.3 for more details) that can intercept messages to and from the simulator, and change their content accordingly, thus simulating depth.

Furthermore, since FSX does not provide with data regarding terrain elevation when underwater, an outside source must be considered. One solution (not yet implemented, but already thought of and tested in a small scale) is to use the Google Elevation API, which can provide with elevation data not only for positive altitudes but also for negative ones, thus allowing for a mapping of the ocean floor to be simulated as close to reality as possible [Google, 2010].

5.1.4 Summary

This section described the central element of the simulation platform – the simulator. First, a method for analyzing flight simulators was introduced, consisting of four main categories – simulation engine, graphics, fault injection and openness. Each of these categories includes several aspects to be considered, when analyzing a simulator. Three simulators (considered to be the most well-known in their area) were analyzed in more detail, as to support the decision process. Microsoft Flight Simulator X was the chosen simulator, after considering its good overall score in all four categories (especially in the openness category, with SimConnect presenting not only a wide range of functions and available variables, but also extensive documentation), along with the structured experiences system (considered to be an important extra feature). FSX provides with a real-time simulation, but also allows for slower- and faster-than-real-time simulations, enabling a more detailed analysis of the simulation and also a more expedite manner for simulating missions (when detail is less important than speed), respectively. SimConnect allows several independent external clients to be connected to the simulator simultaneously, which is suited for an architecture where each individual component has a dedicated connection to the simulator.

After choosing FSX as the simulator, some uses of this simulator in scientific research were presented, focusing on its use as a realistic simulation platform for an emotional assessment and feedback framework (which also reinforced the notion that this simulator is a very realistic one).

Finally, some challenges that arise from using this simulator were introduced, namely how different vehicle type specifications are handled by a simulator with an initially fixed number of vehicles, and also regarding the use of FSX for the simulation of vehicle types other than aircraft (and especially submarines, which is also considered in more detail in section 6.3).

5.2 Control Panel

As mentioned before, the Control Panel is the main interface element between the platform and the user. It allows for the configuration of the system as well as most aspects pertaining the simulation. The Control Panel is divided in five main sections – platform configuration, scenario, teams, disturbances and mission definition. The latter four sections are used for the configuration of each of the respective components, in accordance to the defined dialects, which were fully described in chapter 4.

5.2.1 Platform Configuration

The platform configuration section of the Control Panel allows for the specification of platform-wide definitions, such as the connection to simulator, agent communication platform, network configuration and base logging directory. It also provides with information regarding the state of the remaining four configuration sections, as well as offers a live log of communications and actions.

The simulator section requires only the specification of the IP of the computer running the simulator – this IP is chosen from the entries present in the SimConnect client configuration file[7] [Microsoft Corporation, 2008b]. The simulator test verifies if the simulator is running on the specified IP address. Network configuration requires only the specification of the network nodes that are capable of running the several components of the platform. These computers must have been configured with the necessary client software for the simulator, as well as the various platform components. The network test verifies that all target computers are reachable and all necessary components are correctly installed in each computer. The agent communication platform (AgentService – see section 6.1) configuration requires the specification of three parameters – the IP address and port of the computer where the platform is running, and the platform's name. In addition, the username, password and name of all platform components that require a connection with the agent communication platform should be specified to be validated against the platform user configuration file. The AgentService test verifies that the platform is running on the specified computer, and authentication information for each component is correct. In case any of the three tests fails, a message is presented below the test buttons, identifying the specifics of the problem. The base logging directory is a folder in which the session folders will be created, containing all produced log files – see section 5.5 for more details on the logging mechanisms. All configuration details are loaded from the registry of the machine running the Control Panel, and can also be saved after changes have been made (by pressing the save button). Two applications can also be launched from the Platform Configuration screen – the Monitoring Tool and the Performance Analysis Tool (both described below).

In the bottom half of the panel, the status of the Control Panel can be found – on the left, information regarding all messages sent and received by the Control Panel are displayed (detailing sender/receiver and content); on the right, the status of the remaining four configuration sections

[7] Refer to the SimConnect installation documentation for more details

– Scenario, Team, Disturbances and Mission – is shown, indicating if each configuration file has been loaded to the Control Panel, and if the configurations have been launched into the platform (also, if that implies the creation of ATC or Vehicle Control agents, an indication of how many agents are being created and the progress of the agent creation process is shown). Figure 5.7 shows the platform configuration screen, after the tests have been successfully performed, and scenario and team configurations launched (including the creation of ATC Agents and Vehicle Control Agents).

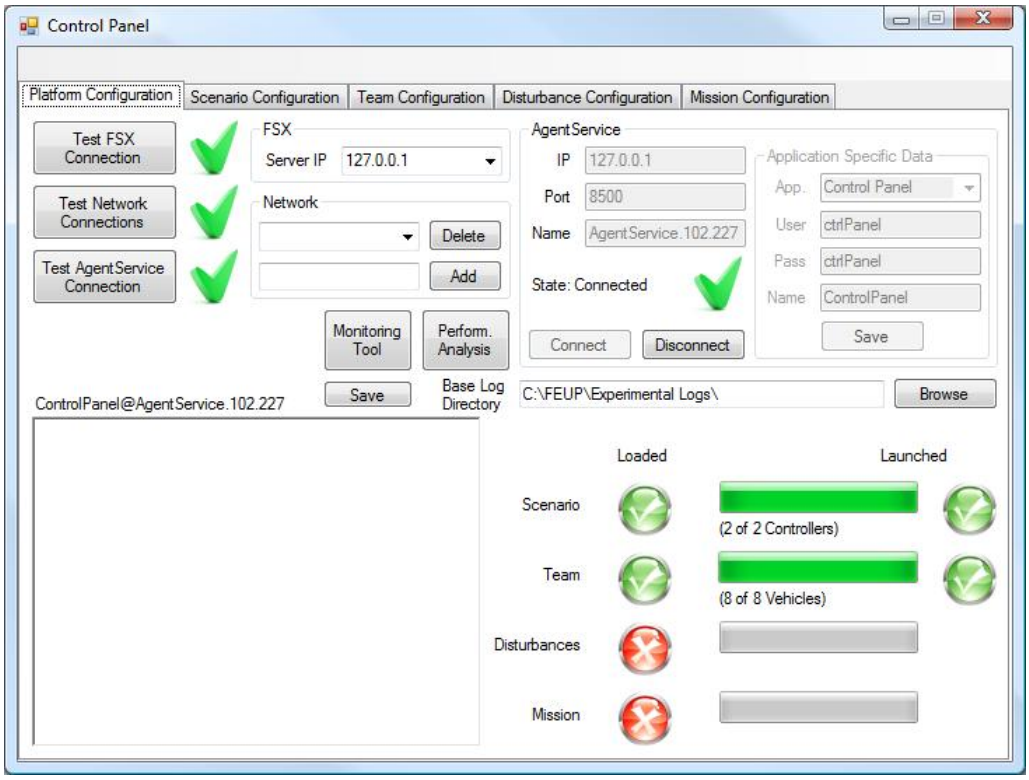


Figure 5.7: Control Panel – Platform Configuration After Launching Scenario and Team

5.2.2 Scenario Configuration

The scenario configuration section of the Control Panel allows the user to open, edit and save scenario description files (specified according to SDL), and to launch the currently loaded scenario to the platform, effectively creating any new vehicle types (see section 5.1.3.1) and creating an ATC Agent for each existing controller. Figure 5.8 shows the main scenario configuration screen of the Control Panel, where four main areas are clearly visible – bases of operations, no fly areas, controllers and agent types –, corresponding to the four elements of the scenario description language (see section 4.2). Due to screen size limitations, and given the amount of information required by each of the four main elements, additional panels are accessible in each of the four main areas for introducing or editing the remaining information. In all these panels, as well as in the configuration of teams, disturbances and missions, the generation of identifiers is performed automatically,

as to abstract the user from such low-level implementation details. Each distinct entity is identified by a letter, unique to that entity, followed by a sequential number (for instance, and as can be seen in Fig. 5.8, bases of operations have identifiers starting with *b*, areas have identifiers starting with *a*, controllers have identifiers starting with *c*, and agent types have identifiers starting with *y*).

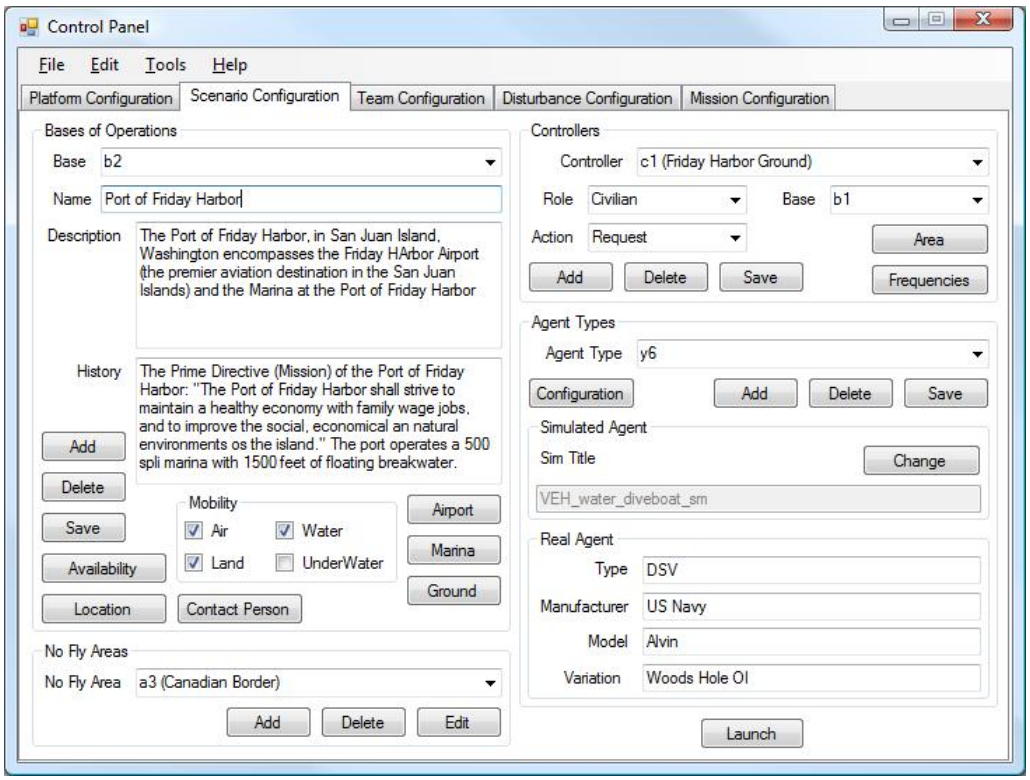


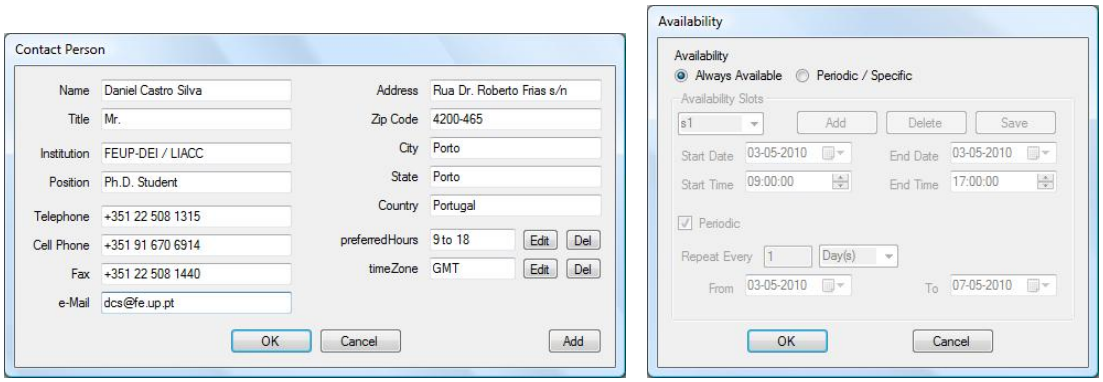
Figure 5.8: Control Panel – Scenario Configuration

5.2.2.1 Bases of Operations

Only a few details of a base of operations are accessible in the main screen – name, description, history and mobility. All other fields are accessible through the respective buttons – location, contact person, availability, and possible airport, marina and ground base. As mentioned before, the airport, marina and ground base are only available depending on the values of the mobility attributes.

Figure 5.9(a) shows the screen for editing the contact person for the base of operations.Given the possibility to add any additional information items to the contact person (such as preferred contact hours, or time zone, as included in Fig. 5.9(a)), the screen size is dynamically adjusted to existing additional information details (the sizes of the additional details' labels are calculated and the largest one is used to determine the appropriate width for the screen, thus guaranteeing that every piece of information can be read in full by the operator).

Figure 5.9(b) shows the screen used to edit the availability of a base of operations. The base can be available at all times (in which case the 'always available' option should be selected), or



(a) Contact Person Screen (b) Availability Screen

Figure 5.9: Contact Person and Availability Screens

only during certain time periods. For each given time period, the specific hours of availability can be specified, as well as the recursion rule for the period (recursion can be specified using the initial and final dates that delimit the recursion, and with a repetition on a daily, weekly, monthly or yearly basis).

Figure 5.10 shows the airport editing screen, which contains several elements that can be directly edited and the buttons to access the screens in which the other details can be edited. The contact person screen is the same as shown above for the base of operations. Simple elements (name, ICAO and IATA codes, description and magnetic variation at the airport location) can be edited directly in upper left part of the screen. All utilities are shown in the bottom left part of the screen, and all types of utilities (tower, water, fuel or battery facilities) can be added by pressing the respective buttons. The right side of the screen allows for helipads and parking spaces to be specified. For more complex elements (runways, taxiway network and hangars), the respective buttons allow for additional panels to be opened, in which the specification of each of these elements is performed.

Two other screens, similar to the one used to configure an airport, exist for the configuration of port and ground base, and their respective characteristics. Again, simpler elements are configurable in the port and ground base panels directly, but more complex elements (such as waterway and quay networks for ports, and the road network for a ground base) have their own configuration screens.

5.2.2.2 No Fly Areas

The definition of the geographical area of a no-fly area is made by accessing the Area Configuration screen, shown in Fig. 5.11. For usability reasons, the denomination of the area is also used on the main interface, along with the area identifier, to help the operator identify the area he wishes to edit or remove.

This definition of the area can be performed either on the left side, using the respective buttons and text fields, or on the right panel, using the provided graphical interface. It can be defined as

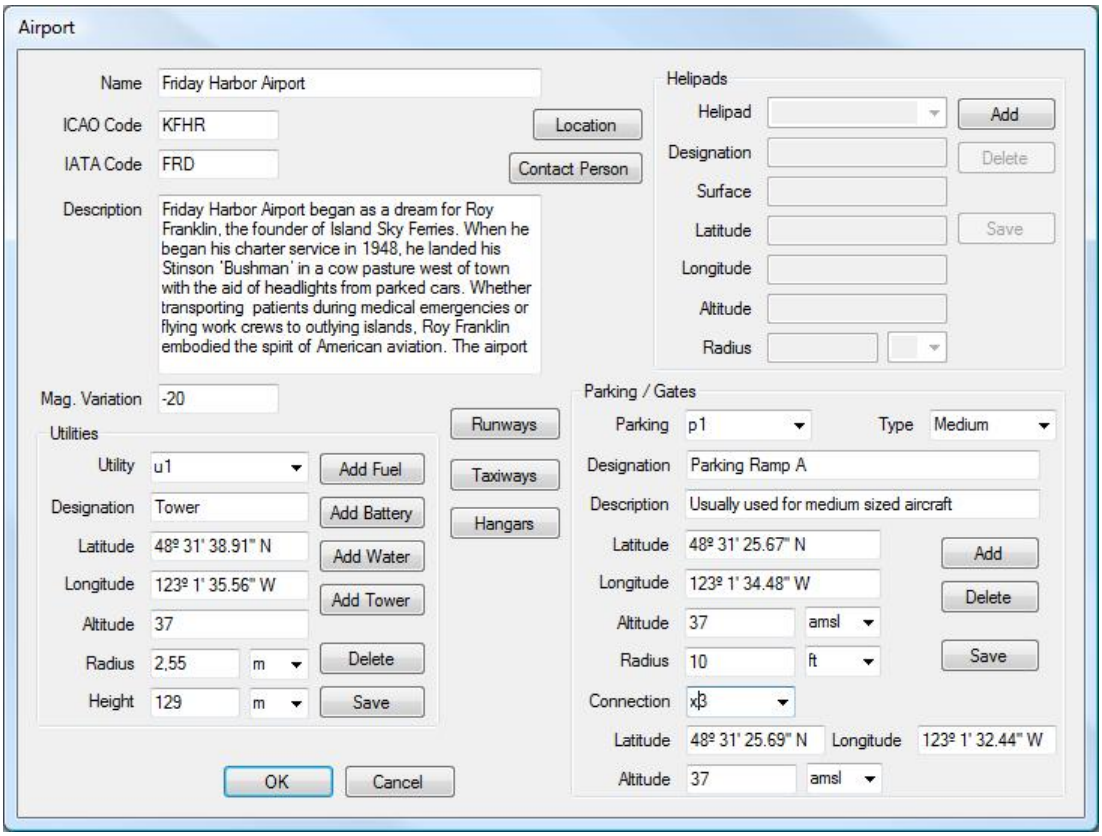


Figure 5.10: Airport Screen

either a polygon or a circle, as mentioned in the definition of the area element (in section 4.2.5). For the configuration of a polygon, it is defined by a list of vertexes (the last one connecting to the first, to close the polygon), each of which can be edited in terms of latitude and longitude. The polygon defined by the vertexes is then extruded vertically, using the minimum and maximum altitudes as delimitors. The graphical interface uses a web browser with a Google Maps plugin[8], allowing for the area to be defined interactively: in the case of a polygon, the several points that define a polygonal area are represented as white squares, and can be freely moved in the map simply by clicking and dragging; also, new vertexes can be interactively added by dragging one of the gray squares between any two consecutive vertexes.

The availability button opens the area availability screen (similar to the one shown in Fig. 5.9(b)), which allows for the specification of the temporal period(s) when the restriction is active.

Additionally, this screen also allows for KML (Keyhole Markup Language, an XML-based language used to display geographical features in browsers and other applications, such as Google Earth [OGC, 2008a]) files representing the desired area to be imported or exported. As such, the desired area can be created using Google Earth or a similar application, and then imported to the application. Conversely, the area can also be exported for use (or tweaking) with other application.

[8]For more information, refer to `http://code.google.com/apis/maps/documentation/javascript/`

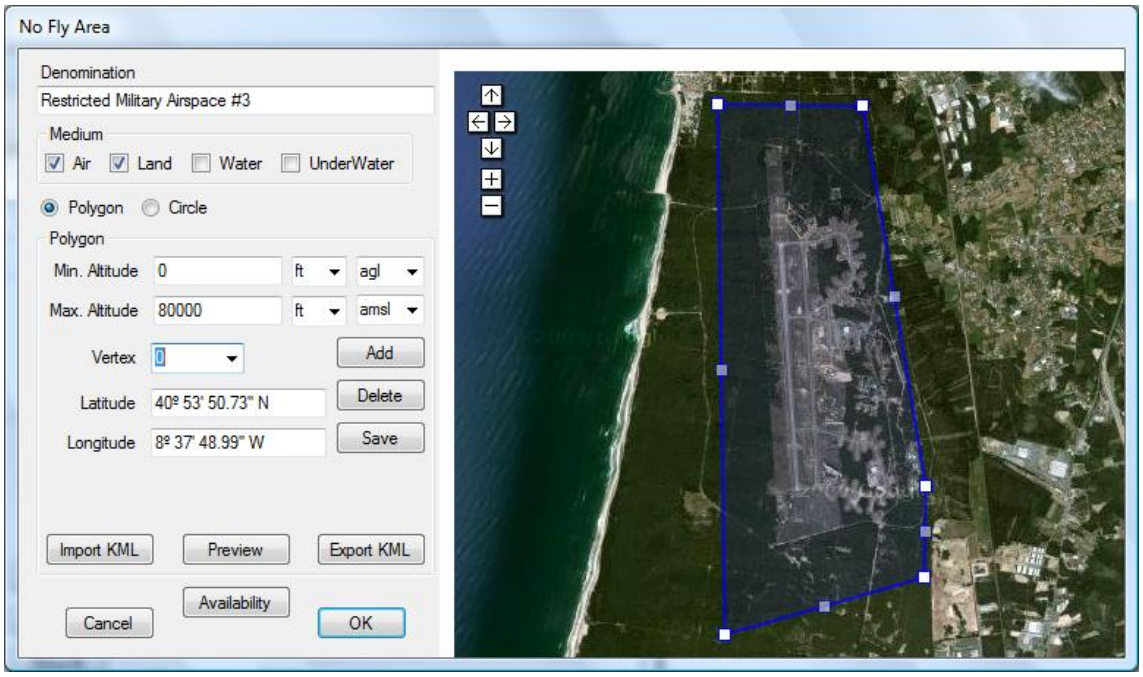


Figure 5.11: Control Panel – Area Configuration

5.2.2.3 Controllers

Part of the information regarding controller configuration (role of the controller, required action by vehicles maneuvering in the area over which the controller has jurisdiction, and base of operations the controller is associated with) is present on the main scenario screen (Fig. 5.8). The details of the area over which the controller has jurisdiction can be specified by pressing the *Area* button on the controller area of the main screen; the controller area screen is identical to the one presented above for no-fly areas, and the denomination of the area is also used to help identify the controller in the main interface. Additionally, the frequencies used by the controller can also be specified in the frequency configuration screen.

5.2.2.4 Agent Types

Two of the five categories of agent type information can be defined in the main scenario screen – simulated agent and real agent. The remaining three categories (physical, performance and payload layout) are configured in the vehicle type configuration screen, accessible by pressing the *Configuration* button. The vehicle type configuration screen presents a different graphical layout depending on the type of vehicle being edited (different vehicle types have different characteristics, as seen on section 4.2.7). Figure 5.12 shows an example of a vehicle type configuration screen, in this case for an aircraft.

Other vehicle type configuration screens present a similar layout, in the sense that physical characteristics are on the left side, while performance characteristics are on the middle, and payload layout configuration is on the right side. Physical characteristics include dimensions, weight

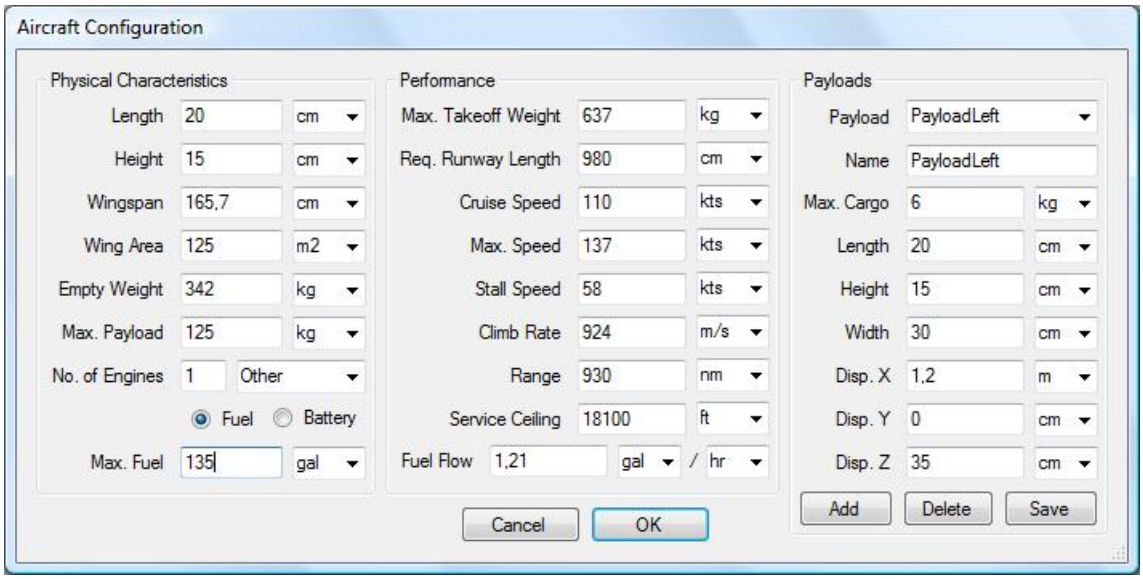


Figure 5.12: Control Panel – Vehicle Type Configuration

of the vehicle and maximum amount of fuel/cargo it can hold. Performance attributes include speeds (normal cruise speed, maximum speed and, in the case of aircraft and submarines, stall speed), fuel (or energy) consumption, and other aspects, such as operational range. The payload configuration area allows for payloads to be added, deleted and edited, and, for each payload, its location relative to the geometric center of the vehicle, size (given by the dimensions of a bounding box), and maximum cargo can be specified.

5.2.3 Team Configuration

The team configuration section of the Control Panel allows the user to open, edit and save team description files (specified according to TDL), and to launch the currently selected team to the platform, creating a Vehicle Control Agent for each existing vehicle in the team. Figure 5.13 shows the main teams configuration screen of the Control Panel, where most of the information regarding a team can be specified.

Simple elements (team name, description, history and purposes, as well as mobility) can be specified directly on the left side of the screen. Contact person information is specified in the same manner as already shown above for a base of operations.

The usable bases of operations can be chosen among existing bases, on the bottom left part of the screen – the left list shows all available bases of operations, and the list on the right corresponds to those bases which can be used by the team. The bottom right part of the screen is dedicated to the specification of additional no-fly areas. The interface and specification is identical to the one presented above for the scenario.

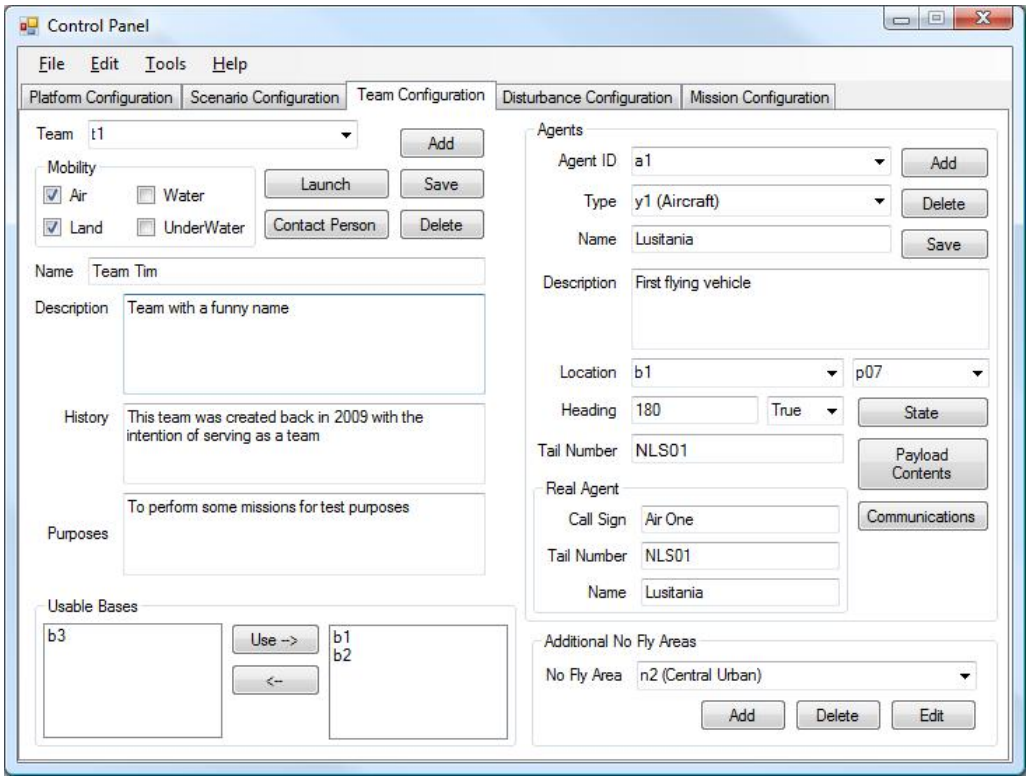
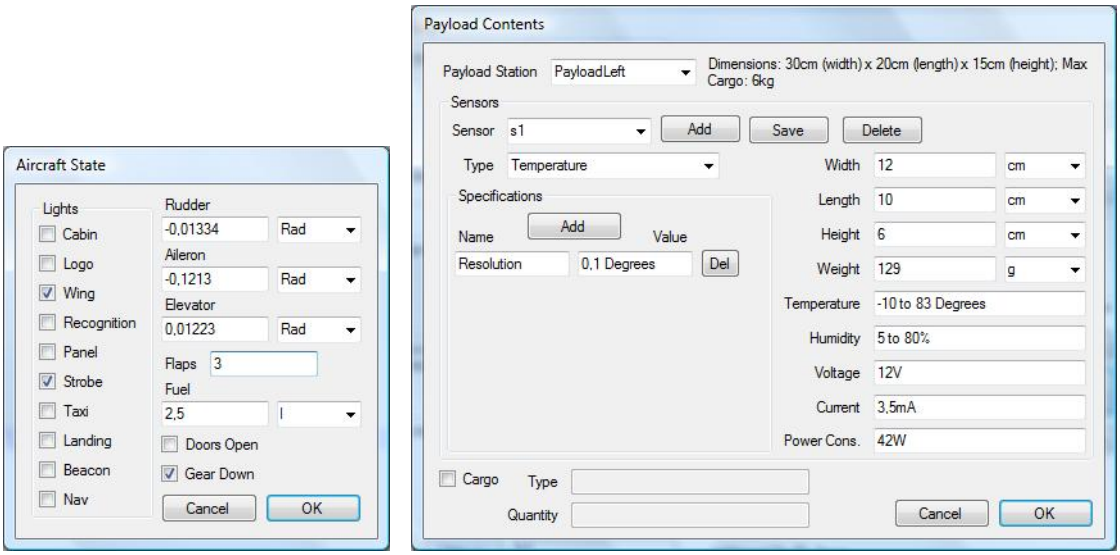


Figure 5.13: Control Panel – Teams Configuration

5.2.3.1 Agents

Agent configuration is performed in the upper right section of the teams configuration screen. Three buttons allow for the configuration of vehicle state, payload contents and communication details in the respective configuration screens. Similarly to what happens with vehicle type configuration, vehicle state configuration screens also differ according to the type of vehicle being configured – Fig. 5.14(a) shows an example of a vehicle state configuration screen, in this case of an aircraft. State information varies according to vehicle type, but generally includes the current amount of fuel the vehicle has (or battery state), the state of the vehicle lights, and the position of control surfaces.

The payload content configuration screen, shown in Fig. 5.14(b), allows for the configuration of the contents of each payload station – sensors and cargo. When a payload is selected, its characteristics are displayed as to help the operator manage the contents of the payload (for instance, by not exceeding the maximum weight allowed for each payload). Several sensors can be added to each payload, each with several possible features – along with the typical features, presented on the right, such as dimensions, weight and some operational characteristics, new specification details can be added. The payload may also contain cargo instead/in addition to the sensors, in which case the type and quantity of the cargo should be indicated.



(a) Aircraft State Configuration Screen (b) Payload Content Configuration Screen

Figure 5.14: Aircraft State and Payload Contents Configuration Screen

5.2.4 Disturbances Configuration

The disturbances configuration section of the Control Panel allows the user to open, edit and save disturbance description files (specified according to DDL), and to launch the disturbances to the platform, effectively creating the disturbances manager, and providing it with information regarding all specified disturbances. Figure 5.15 shows the main disturbances configuration screen of the Control Panel, where the denomination and type of each disturbance can be specified on the upper left part of the screen.

The initial location of the disturbance can be configured directly on the main disturbances screen, on the middle left part of the screen. It can be a specific location or the location can be generated in a random location, within a specified area (in which case, the area is specified in the same manner as with no-fly areas, already shown above). The configuration of the temporal availability of the disturbance is also performed directly on the bottom left part of the screen – this specifies when the disturbance first appears, and when it ceases to exist. The disturbance may begin its existence at a specific point in time, or it can appear at a random moment, during a specified time frame. The end of the disturbance may be specified as its beginning, or it can be left as unspecified, in case there is no predetermined time frame in which it should cease to exist (as is the case of a fire, which disappears only by natural causes, such as rain, or by the action of the team of vehicles).

Mobility configuration, seen on the bottom right part of the screen, includes the definition of the type of mobility (see section 4.4.3 for the description of the available mobility types) and several configuration parameters, according to mobility type. Some of these parameters, such as heading or random variation and distribution can be configured directly on the mobility section of the disturbances screen, while the details on other parameters, such as speed, the motion path (in

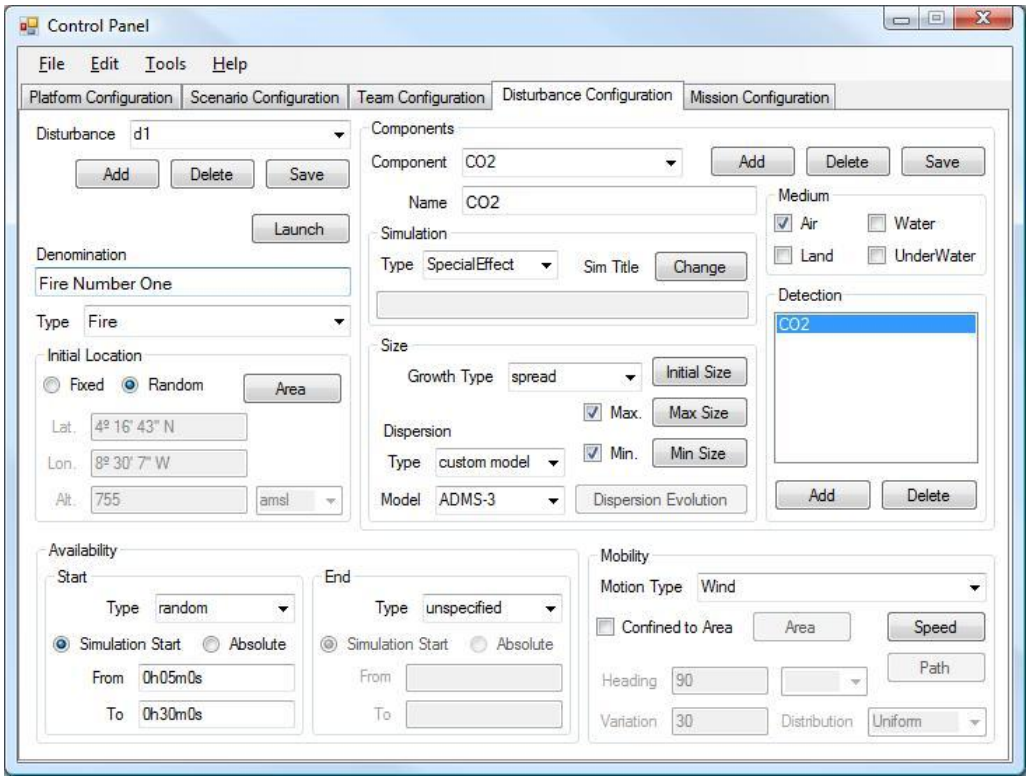


Figure 5.15: Control Panel – Disturbances Configuration

case of a path mobility), and the confinement area (in case the disturbance is confined to a given area) are specified on separate screens.

Component configuration, available on the upper right part of the screen, allows for the specification of the components that comprise each disturbance. For each component, four major option groups are available: simulation, medium, detection and size. In the simulation group, a visual and/or physical equivalent of the component may be specified within the simulator – this allows for a visual feedback of the disturbance and, if possible, the simulator will handle the simulation of the specific component, which may be sensed by the Vehicle Control Agents. This visual/physical representation is selected in a new panel from all available objects of a given type (such as visual effect, vehicle, or other object type) within the simulator. In the medium group, the type of vehicles that can perceive the component can be specified. Related to this group, the detection group specifies the sensors a vehicle needs to be equipped with, in order to detect the specific component. Finally, in the size group, the size evolution of the component, if any, can be specified. Possible minimum and maximum sizes can be specified, in case the size of the component is not constant and such size limits exist. Alternatively, a dispersion pattern can be used, in which case either an existing dispersion model can be used, or the dispersion parameters can be specified. Both dispersion and size evolution specifications are performed on a different screen, as are the specifications of maximum and minimum sizes.

5.2.5 Mission Configuration

The mission configuration section of the Control Panel allows the user to open, edit and save mission description files (specified according to MDL), and to launch the mission to the platform, so that the vehicle control agents that compose a team can plan the execution of the specified mission. Figure 5.16 shows the main mission configuration screen of the Control Panel, where the denomination and description can be specified on the upper part of the screen. Also, a team can be specified (in case multiple teams are created, a mission can be sent to a specific team).

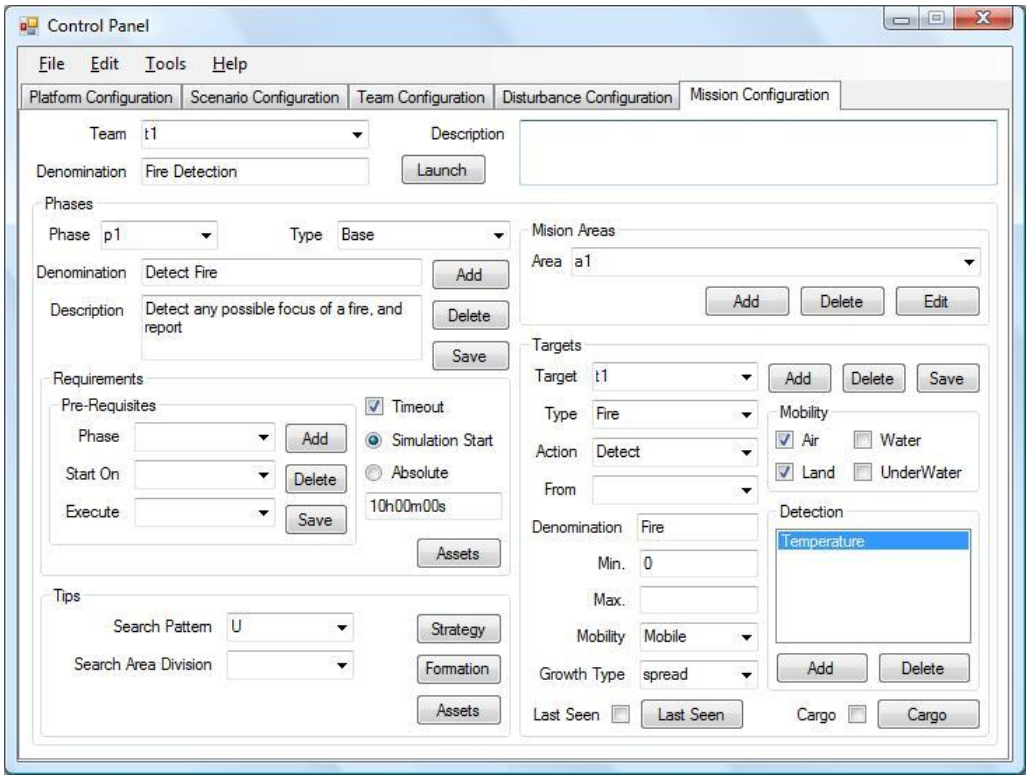


Figure 5.16: Control Panel – Mission Configuration

The bottom part of the screen is dedicated to the configuration of mission phases. For each phase, mission areas, requirements, tips and targets can be configured in each of the respective configuration parts of the screen, and the denomination, description and type of phase can also be specified on the upper left part of the phases screen. The configuration of mission areas is similar to the configuration of no-fly areas in the scenario configuration section (see section 5.2.2.2) and defines the geographical areas where the mission will take place. The requirements for each phase define the hard constraints that have to be met. Requirements include a timeout, that specifies how long the team has to complete the mission; the assets that must be used (asset types that can be specified include vehicle types, sensors and/or cargo); and the mission phases that must be complete before starting the current phase. In case there are phase dependencies, the current phase can be defined to start once one or all the goals of the previous phase have been met, and if it should be executed only once or each time the goal of the previous phase is achieved. The tips

represent soft constraints, that the operator can introduce in an attempt to steer mission execution. Tips include assets (which have the same definition as in requirements), search pattern, search area division, formation and strategy. The search pattern and area division can be used when the phase has a search component (as is the case of detection), and allow the operator to specify the search pattern to be used (which include parallel, sector, trackline, as well as inward or outward circular or square patterns, among other), and if the mission area should be divided among the several agents and how this division should be made. The formation tip allows for the specification of the physical formation the vehicles should adopt (which can be defined using either a leader and follower approach, or a grid occupancy approach, as seen in section 4.5.2). The strategy defines a set of tactics to be used by the team, with their respective triggers and actions. The target definition area allows for the specification of target multiplicity, mobility, and the required sensors to detect it. It is possible to specify whether or not the target has been previously seen and, in case it has, the location, as well as heading and speed, if the target is mobile. Also, it is possible to link the target to one from a previous phase in the mission, if it is inherited from a previous phase.

5.2.6 Summary

This section detailed the Control Panel as the main interface component between the developed platform and the operator. The five configuration sections were presented - platform, scenario, team, disturbances and missions (the four latter ones being a direct representation of the four languages presented in the previous chapter). The first configuration section (platform) allows the operator to configure some platform-wide definitions, namely the connection to both simulator and agent communication platform, and also to monitor the global status of the platform. The scenario configuration allows the operator to specify existing bases of operations, controllers, and vehicle types, as well as global no-fly areas. In the team configuration section, the operator specifies the composition of the team, detailing each existing vehicle, and also the bases of operations the team can use and additional team-specific no-fly areas. The disturbances section allows for the configuration of several disturbances, which includes the specification of disturbance location, availability, mobility and also the configuration of each of the disturbance's components. Finally, mission specification is performed in the mission section – for each phase of the mission, requirements and tips can be specified, as well as the mission areas and targets.

Several aspects were taken into account when developing this component, as to provide the operator with an interface with good usability. These aspects include user control (the parameters the operator can change are limited at each moment), consistency throughout the several screens (also, groups of parameters are used to easily identify the location of certain parameters), error prevention (most input fields are limited to a list of allowed values, and several verifications are made for other fields), or the promotion of recognition over recall (by showing, for instance, the area name along with its identifier, or the characteristics of a payload when configuring its contents). Also, the use of visual, interactive methods to specify certain elements, as is the case of the use of Google Maps to specify an area, helps promote a better usability experience.

Some developments have been identified to be implemented in the future, as to further improve usability. One such feature, in order to increase flexibility, would be to provide accelerators, both for accessing menu actions (other than actions over files), and also to automatically focus on specific fields (this is likely to decrease the time experienced users spend configuring the several aspects of the simulation). Another possible development would be to extend the use of the visual editing capabilities (used in the specification of areas) to other elements with a geographical nature, such as airport, port and ground base. One other possible future development is the application of templates, to help decrease specification time, by automatically filling several fields with their default and/or most typical values for certain elements.

5.3 Disturbances Manager

The Disturbances Manager is responsible for managing all disturbances within the environment (see section 4.4 for disturbance specification), when the simulator is unable to do so by itself. When the disturbances are loaded into the environment (see section 3.2.2), these are effectively sent to the Disturbances Manager, which will then handle all interactions between vehicles and disturbances, when they cannot be sensed directly through the simulation environment. Since each disturbance may contain any number of components, each of which with a different representation within the simulator (or no representation at all), disturbances have to be broken down into those components.

Components that can be simulated and sensed through the simulation environment are created in the simulator, according to the specifications (step 1 in Fig. 5.17(a)) – these are usually stationary and devoid of a growth or dispersion pattern. These components are monitored periodically by the Disturbances Manager, as to assure that their evolution matches the defined parameters; if necessary, the components within the simulator are modified to match the specification. These components are sensed directly by the Vehicle Control Agents by using the available simulator capabilities (step 3 in Fig. 5.17(a)).

When the simulator cannot simulate a specific component, or it cannot be sensed by the vehicles via the simulator, it has to be simulated by the Disturbances Manager. For each component, its location and size are maintained and frequently updated to match the definition.In some cases, the simulator can provide with a visual and physical simulation of the component, but not with the means for it to be detected by the Vehicle Control Agent. In these cases, the Disturbances Manager delegates part of the simulation to the simulator, but still needs to handle communications with the Vehicle Control Agent regarding the component.

In order to handle the communications between disturbances and vehicles, the Disturbances Manager needs to be aware of the sensing capabilities and location of each vehicle. The sensing capabilities of the vehicles are gathered via the agent communication platform, where all vehicles publish their information (see section 6.1). The location of the vehicles has to be read from the simulator. An exception to this is the case of underwater vehicles – since the simulator is unable to simulate depth (which is simulated by the Vehicle Control Agent – see section 6.3.2.1), the

Disturbances Manager needs to ask the Vehicle Control Agent for this information. As to maintain communication requirements to a minimum, this request is only performed when the location of the vehicle coincides (in latitude and longitude) with one or more disturbance components – the depth is then used to determine whether or not the vehicle can actually sense the component, and, if so, the value that should be sent to the vehicle. Also with the intent of decreasing communication requirements, vehicle locations may be read at different time intervals – by taking the vehicle's current location, heading and speed into account, as well as bounding boxes for all disturbance components, the Disturbances Manager determines, for each vehicle, whether or not it will be within sensing range of any disturbance in the near future; in case a given vehicle's location and motion does not place it close to a disturbance in the near future, the temporal hiatus before the next request for information is larger, and it decreases as the vehicle moves closer to the sensing range of a disturbance. When the vehicle can sense the disturbance, the rate at which the Disturbances Manager requests for information is determined by the sensors the vehicle carries and the frequency at which they collection data (for instance, for a continuous sensor, information regarding the vehicle location is requested every simulation cycle, and data is also sent to the vehicle every cycle; on the other hand, for sensors that can only register a new value twice per second, for instance, information is only requested and data sent at that rate).

Algorithm 1 illustrates how vehicles are associated to disturbance components, in order to decrease computational requirements for the Disturbances Manager. This process is performed on a regular basis, to account for vehicles entering or leaving the simulation. Algorithm 2 shows the main process of the Disturbances Manager, that determines which vehicles, if any, can sense each of the disturbances' components, and send a message to such vehicles, with the appropriate values with the simulated sensor readings.

Algorithm 1 Vehicle Update Algorithm

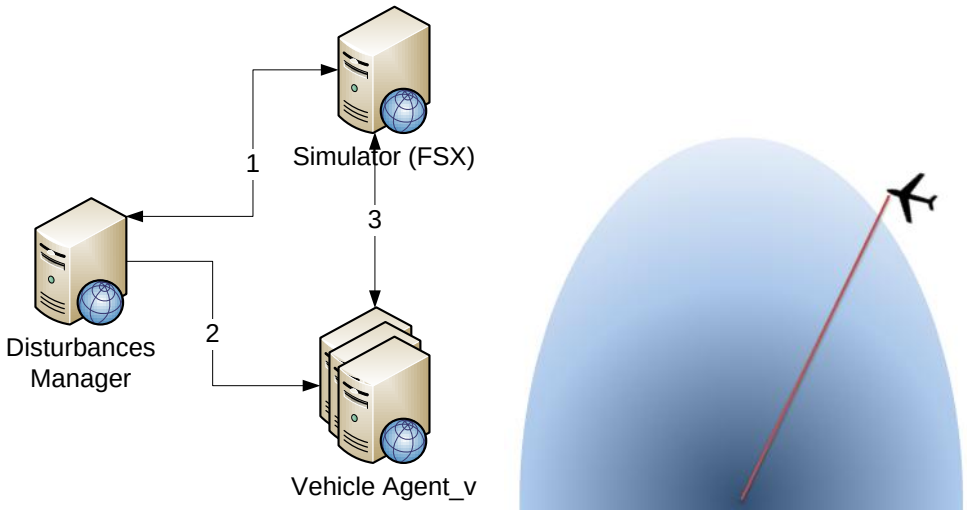
```
loop
  vehicles ← getVehiclesFromAgentService()
  for all disturbance d in disturbances do
    for all component c in d.components do
      for all vehicle v in vehicles do
        sensors ← v.getSensors()
        if detects(sensors, c) then
          c.vehicles.Add(v)
        end if
      end for
    end for
  end for
end loop
```

Whenever a vehicle is perceived to be within sensing range of a disturbance (see Fig. 5.17(b)), a message is sent to that vehicle, containing the simulated sensor readings (step 2 in Fig. 5.17(a)). This message is interpreted by the vehicle as the sensor values and these are used to determine disturbance location, concentration, or other property, depending on the current mission goals

Algorithm 2 Disturbance Detection Algorithm

```
loop
  vehicles ← getVehiclesFromFSX()
  for all disturbance d in disturbances do
    for all component c in d.components do
      for all vehicle v in c.vehicles do
        if distance(c, vehicles.getVehicle(v).Position) ≤ c.radius then
          sendMessage(v, determineValue(c, vehicles.getVehicle(v).Position))
        end if
      end for
    end for
  end for
end loop
```

and disturbance type. The example shown in Fig. 5.17(b) can be interpreted as a source for a polluting chemical with a simple linear gradient dispersion model; when the represented aircraft is at a distance from the source of the chemical that is already within the dispersion area and the sensors aboard the aircraft are able to detect the presence of that chemical, a message is sent from the Disturbances Manager to the aircraft's control agent, containing the concentration of the chemical at its current location. While the vehicle remains within the chemical dispersion area, the Disturbances Manager keeps sending these messages, at a rate consistent with the aircraft's sensor's frequency of operation, as seen above.



(a) Communications (b) Distance Evaluation

Figure 5.17: Disturbances Manager Communications and Distance Evaluation

Additional interactions with the simulator are required for certain disturbances:

- In the case of a disturbance that has a motion pattern dependent of wind direction, the Disturbances Manager needs to communicate with the simulator in order to determine wind direction and speed at the location of the disturbance, to determine the appropriate motion.

- When the disturbance is located on the ground and is mobile, the Disturbances Manager also needs to communicate with the simulator to determine the ground altitude at the location of the disturbance. A particular case also occurs when the disturbance only exists on land, but not on water, in which case the Disturbances Manager also requests information regarding the surface type.

The current implementation of the Disturbances Manager considers only static disturbances with fixed size, and only two simple models for detection – fixed area and a linear model for concentrations. Even though this implementation does not meet with the entire specification for disturbances (see section 4.4), it is representative of simple disturbances, and allowed for a validation of the approach. For that, two disturbances were created at two fixed locations, and an aircraft (carrying sensors that could be used to detect the components of the two disturbances) was place in a route that would intersect the area where the disturbances were active. The Disturbances Manager, via the agent communications platform, determined that the aircraft possessed the required sensors to detect the disturbances, and during the time the aircraft was within the area where the each component could be sensed, sent messages to the respective Vehicle Control Agent, containing the detected values. The results were not perfect at first, since a slight deviation existed between the theoretical values (as simulated by the Disturbances Manager) and the values reported by the Vehicle Control Agent. This deviation was of a spacial nature, and even though it went unnoticeable for the first disturbance (fixed area and sensor value), in the second disturbance, slight variations in the coordinates for each sensor reading were detected. This problem can be explained by the time it takes from the moment the Disturbances Manager receives the information regarding the location of the vehicle to the moment the Vehicle Control Agent receives the message from the Disturbances Manager containing the registered sensor value – during this time, the vehicle continued moving and hence the coordinates associated with the registered sensor value were slightly different from the intended ones. This problem has been corrected by including the coordinates of the sensor reading in the message sent by the Disturbances Manager. This allowed the Vehicle Control Agent to associate the received value with the correct location.

One development that could be implemented in the future would be to provide this component with a graphical interface that allows the operator to monitor the state of each disturbance.

As mentioned in section 4.4.4, there are several existing dispersion models at use nowadays by several countries and independent institutions. These models could be included in the Disturbances Manager as dispersion modules, to be used by each disturbance component according to the specification. These modules are to be registered at a system level, so that when a model is not available as a module, the respective option is removed from the control panel, when specifying the dispersion model.

In order to provide a more realistic simulation of certain disturbances, a more detailed simulation for those disturbances could be implemented. For instance, in the case of a fire, it produces different gases at different concentrations, and burns at lower or higher temperatures depending on the materials that fuel the fire. Some information could be used, such as the surface type and condition, to provide with a more realistic model for fire – FSX features twenty-five different surface

types, and four surface conditions (normal, wet, icy and snow), which influence the characteristics of the fire. Also, the direction of the wind, allied with terrain morphology can determine the direction and rate at which the fire spreads. The terrain morphology can also be used to provide with a more realistic velocity representation for a vehicle or person – when the road has a positive slope, the velocity at which the vehicle or person travels is likely to be lower than when the road has a negative slope; additionally, if the road has several turns, the velocity is also likely to be lower than when traveling in a straight line. The road surface type and conditions can also contribute to a better approximation to what the vehicle or person's velocity would be like in reality.

5.4 Monitoring Tool

The Monitoring Tool allows for both a macro and micro real-time analysis of the state of the simulation. It provides a real-time visual feedback of the state of the simulation on the main screen, and also allows for a flexible textual feedback on the state of each individual vehicle that comprises the team to be shown on a dedicated screen within this tool.

5.4.1 Simulation Status Monitoring

The Simulation Status Monitoring tab provides with a system-wide visual feedback of the state of the simulation, as well as a list of vehicles that can be monitored individually – see Fig. 5.18.

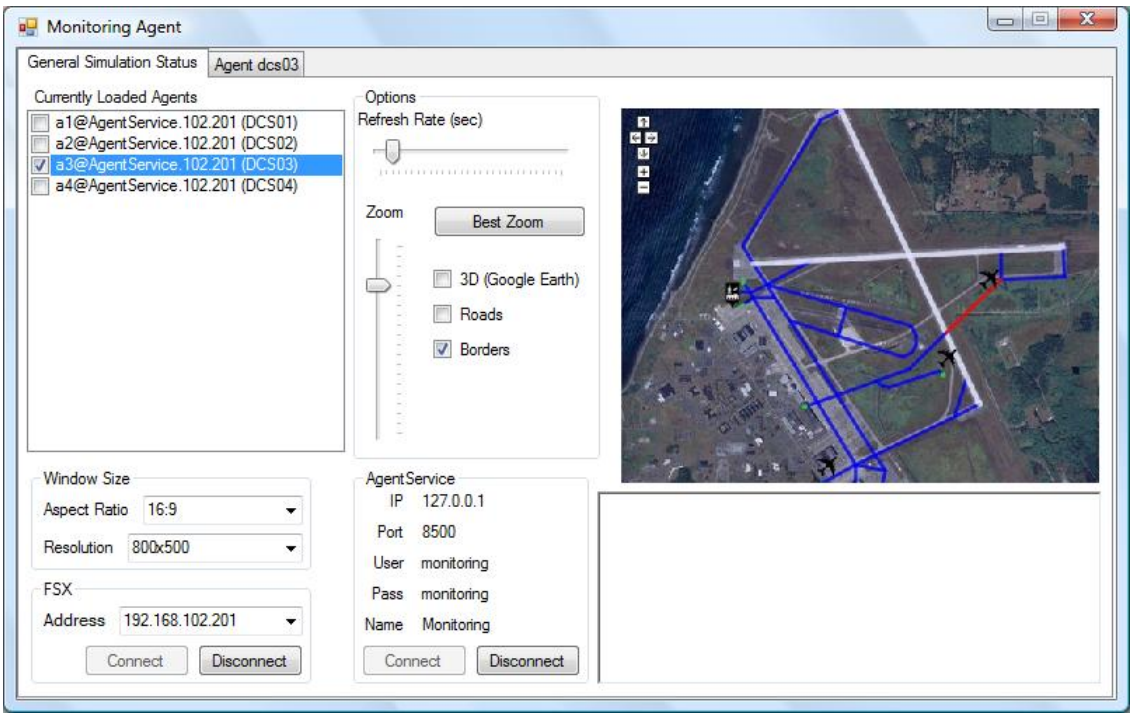


Figure 5.18: Monitoring Tool – Simulation Status

On the left side of the screen, a list of all team vehicles is shown, allowing the operator to choose the ones he wants to monitor in more detail. By checking the square next to the name of

a vehicle, a new tab is created, where several variables regarding the vehicle can be monitored in real-time – see section 5.4.2 below. On the left bottom part of the screen, information regarding the connection to both simulator and agent communication platform is shown. Also, a window size configuration area allows for the operator to modify the size of the window, allowing for more space to be used in visualizing the information – several predefined aspect ratios and dimensions are allowed to be chosen, and the several components are resized and relocated, as to better fit the new screen dimensions. On the right bottom part of the screen, all messages sent and received by the Monitoring Tool are shown. Finally, on the top right section of the screen, a map is shown, containing all structures (airports, ports and ground bases from the bases of operations), as well as vehicles in the simulation. As was the case in the Control Panel, for area specification, a Google Maps plugin is used within a web browser, that allows for the operator to interact with the map, positioning it to the desired location. Some other parameters can be adjusted from the available controls outside the map, such as zoom level (which can also be manipulated within the map), whether or not the map should include the representation of country borders or roads, if the map should be represented in two or three dimensions, and the rate at which the location of the vehicles within the map should be refreshed. Finally, the 'Best Zoom' button adjusts the location and zoom level of the map so that all vehicles and structures stay within the represented area.

This screen, and especially the map, provides a visual manner in which the operator can rapidly perceive the state of the simulation, at a global level, by assessing the current location of all team vehicles. One improvement that could be introduced to this screen (and which is also related to a future development that could be implemented in the Disturbances Manager presented in the previous section) is to include a visual representation of the disturbances that exist in the environment. This would allow the operator to have a better understanding of vehicle movements, and the relation between these movements and existing disturbances.

5.4.2 Vehicle Status Monitoring

Individual vehicle status can be assessed by a number of state variables, available for real-time monitoring. These variables are grouped into categories, and the operator can choose which categories to monitor, and which variables within each category to see. Categories are represented as boxes that can be freely moved within the monitoring screen, thus allowing each operator to completely customize the display to his preferences, and even with different layouts for different vehicles. Figure 5.19 shows an example of the vehicle status monitoring screen, with four groups of variables being shown, and only one of them receiving data.

One group, named Control Panel (not to be confused with the Control Panel described in section 5.2), is permanently visible, allowing the operator to configure which other groups are visible in the interface, as well as the variables to be shown within each group. The layout configuration of both the control panel and the other groups can also be adjusted from the control panel, to provide a more horizontal or vertical physical arrangement of the variables inside each group – Fig. 5.19 shows the ATC group with a vertical disposition of variables, while the Position/Attitude group has a more horizontal layout. Also, the control panel allows the operator to select only one

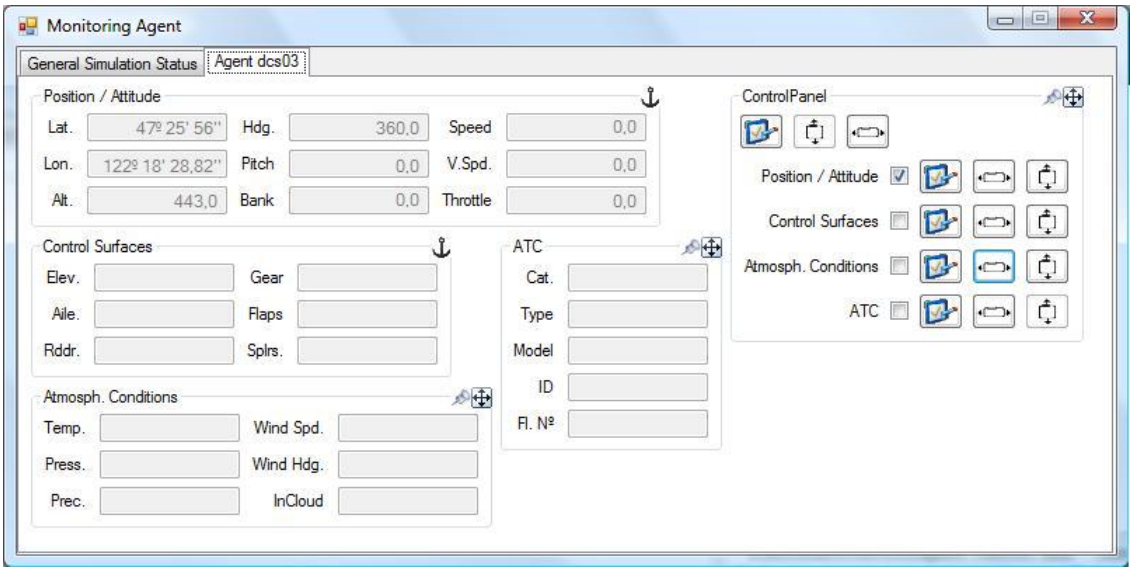


Figure 5.19: Monitoring Tool – Vehicle Status

or some of the groups currently visible to be updated in real-time, with the values retrieved from the simulator (this allows for a reduction of the communication requirements for the Monitoring Tool).

All groups can be freely moved within the panel, simply by dragging them to the desired location. The groups can also be anchored to a specific location (in Fig. 5.19, both the Position/Attitude and the Control Surfaces groups are anchored, while the three remaining groups can be freely moved). When a group is anchored, no other group can occupy the same location, thus somewhat limiting group movements (especially when using a small-sized window for the Monitoring Tool). When not anchored, the groups flow to another location when one group is dragged onto them.

Several categories, each with several variables, targeting aircraft vehicles, have been incorporated into the Monitoring Tool. These categories and variables are listed in Table 5.3. It should be noticed that the categories and variables presented herein are those presented for aircraft; for other vehicle types, some of the variables are replaced by the adequate counterparts for the remaining vehicle types. For instance, the Position and Attitude group retains the same number of variables for underwater vehicles, simply replacing the altitude variable for the depth of the vehicle, but several of the variables are not included for water and land vehicles (some variables, such as vertical speed, are not used in these types of vehicles).

Additional groups of variables can also be added, simply by including their configuration in the application – an example of a new group is related to fault tolerance, and includes a number of variables that correspond to the vehicle's self assessment for failures in a number of systems and instruments (see section 7.2).

Categories	Variables	Categories	Variables
Aircraft Control Surfaces	Elevator Aileron Rudder Elevator Trim Aileron Trim Rudder Trim Flaps Spoilers Landing Gear Brakes	Lights	Strobe Taxi Panel Landing Beacon Nav Logo Wing Recognition Cabin
Position & Attitude	Latitude Longitude Altitude Heading Bank Pitch Speed Vertical Speed Throttle	Environmental Variables	Temperature Pressure Density Wind Speed Wind Direction Precipitation Visibility In Cloud?
ATC	Type Model Airline Flight Number	Ground	Surface Elevation Surface Type Surface Condition Magnetic Variation
Payloads	Number of Payloads	Payload	Payload Weight Content Type Content Count

Table 5.3: Variables and Categories for Aircraft Vehicle Monitoring

One improvement that could be introduced to the Monitoring Tool is to consider layout templates. These layout templates could be defined by the operators, and include the definition of which groups are visible, which variables are visible within each group, which groups are receiving data, and the location and arrangement (vertical/horizontal) of each group. These templates could be applied according to vehicle type, or in a more flexible manner, according to the operator's specification, and would contribute to a higher degree of customization of the interface, thus improving usability.

5.5 Logging Tool

The Logging Tool is responsible for creating log files for all entities involved in the simulation. The logging ability, which was initially concentrated in one application named Logging Tool (hence the legacy name), is actually spread through all components participating in the simulation (even though it can be logically seen as a single service), each one creating a file (or a set of files) for each simulation session, in which every action taken, and every messages sent or received are recorded. These files allow for both a simulation replay, and act as an input to the performance analysis tool (described in the following section).

When the scenario file is launched (step 2 in Fig. 3.10), and before creating the ATC Agents, the Control Panel creates a folder within the platform's base logging directory (which is a parameter of the platform's configuration included in the Control Panel – see section 5.2.1). This folder is identified by the date and time of the simulation session and will contain all generated files for that session. Four folders are created within this session folder – configuration, controllers, agents and disturbances.

- The configuration folder will contain the four configuration files for the session – scenario, teams, disturbances and mission. Each of these files is copied into the configuration folder when the respective component is launched from the Control Panel (steps 2, 5, 8 and 11 in Fig. 3.10).
- The controllers folder will contain one log file per ATC Agent, containing all sent and received messages, as well as the events and decisions made by these agents.
- The agents folder will contain four files per Vehicle Control Agent – state, environment, events and messages. The state log contains a record of the vehicle's state throughout the simulation (containing the position and attitude of the vehicle, as well as many other state variables, such as lights, control surfaces or payload status); the environment log contains a record of the environmental conditions surrounding the vehicle (mainly the variables shown in Table 5.3, in the category Environmental Variables); the events log contains a record of all decisions made and actions taken by the vehicle control agent; the messages log contains all messages sent and received by the agent.
- The disturbances folder contains one file per disturbance, containing a record of the evolution of the disturbance throughout the simulation.

Even though the separation of a mission log over several files in several folders can be viewed as an additional factor of complexity, it also allows for the separate analysis of each element – for instance, for a visual recreation of the mission, only the state (and possibly the environment) files of the vehicle control agents are needed.

The visual playback of the simulation session can be done using the chosen simulator. All vehicle motions and actions, along with disturbances with a visual representation can be recreated according to the log files. However, not all aspects of the simulation can be recreated to detail, namely some environmental conditions.

The frequency at which some data is recorded has an influence on the size of the files that are generated. The most visible example is the case of the state and environment log files for the Vehicle Control Agent – currently, state and environment information is saved once per second; even though it could be desirable to have more detailed data (and state and environment information saved at the best possible rate), the vehicle's path can be fairly represented using this information. On the other hand, saving data more often would translate into an increased effort on several components – all Vehicle Control Agents would request such information from the simulator every simulation cycle (the communication demands could become very high, depending on team

size), and they would have to send that information to the respective log files (also increasing the workload for the file system). This increased strain on the system contributes to a decline in the simulator's performance. Further studies need to be conducted using different team sizes, different frequencies for data retrieval and logging, and also computers with different capabilities (in terms of processor and hard disk speeds), in order to determine the best balance point for the frequency at which data should be logged, and to determine if lower computer capabilities and/or larger team sizes should influence that frequency.

As seen in section 6.3.1.6, the Logging Tool allows for vehicle state files to be converted into other formats, namely the CSV and KML formats, that can then be interpreted by external applications for vehicle path analysis. The CSV format can be read by many applications; it has been used in Microsoft Excel to generate three-dimensional representations of the vehicle paths, as seen in Fig. 6.16(a) and 6.16(b). The KML format can be read by Google Earth and some other geographic software (see Fig. 6.15(a) and 6.15(b)).

5.6 Performance Analysis Tool

The Performance Analysis Tool can only be used after the end of a simulation session, and is used to evaluate one or more sessions, according to a set of performance metrics [Santos, 2010].

Performance analysis, as described herein, does not focus on the stability, scalability or performance of a multi-agent system in general [Lee et al., 1998], but rather on the performance of the team when conducting a given mission.

The missions performed by the team can be classified in several types (as briefly presented in section 4.5.3):

- **Detect**. These missions are those where the vehicles should search for a given target, such as a person, a vehicle, a fire, or any other entity. It is usually mapped as the first phase of a larger mission. The search pattern tip on a mission description file (see section 4.5.2) can be used in search mission phases to specify the preferred method of searching for the target.

- **Detect Origin**. These missions are those where the vehicles should detect the point of origin of some anomaly, such as the source of a pollution cloud, or the hydrothermal vent from where certain chemicals are being released. Some strategies, such as a gradient search can be used to pinpoint the source of a given detectable substance.

- **Measure**. These missions consist in using the vehicles' sensors to detect and measure some substance, such as a chemical, and to create a map with the concentrations of such substance in each measured location.

- **Follow**. These missions are widely used in a military or law enforcement context, when a target, such as a vehicle or a person, needs to be followed to either determine its final location or to track its movements over a given time frame.

- **Load/Unload**. These missions are usually used in combination with other mission types, such as a Detect phase. It consists of having the vehicle drop all or part of the cargo it contains (such as water or fire retardant over a fire), or picking up some additional cargo.

- **Transport**. This mission is an aggregation of two missions of the previous type – it consists in loading a cargo at a certain location and unloading it at another location. It differs from the Load/Unload mission type in the sense that both locations are important, while in the Load/Unload mission type, only the location of one of the actions is important – for instance, in the case of water being dropped on a fire, it is not important to specify the location at which the vehicle loads the water (it can do so in any usable base of operations), but if an aid package is being transported to a refugee camp it is important to specify both origin and destination of the operation.

Performance evaluation is closely related to the type of mission being carried out by the vehicles. As an example, in a transport mission (where the vehicles should transport a given cargo from point A to point B), the most important factors are likely to be the time it takes to transport the cargo and the total amount of fuel spent. In other mission types, such as the search for a fire over a given area, time is not likely to be an important factor, as probably is, for instance, the total area covered by the vehicles. As such, and in order to facilitate the evaluation process, several profiles can be created, each profile corresponding to a different perspective over the simulation, as specified by the operator. In practice, each profile will contain a number of metrics, and their relative weight for the overall performance measurement.

To evaluate the performance of a vehicle or team, a set of metrics, which can easily be extended, was created. Some of these metrics are defined at a vehicle level, while others are defined at a team level. Vehicle-level metrics include fuel consumption, distance traveled, average speed, useful mission time, among others. Team-level metrics include aggregated fuel consumption, total area covered and other measures that consider all vehicles involved in the simulation. Most of these metrics can be displayed in a graphic, reporting the evolution of the metric throughout the simulation – for instance, showing the speed of a vehicle, or the total area covered by the team during the simulation. The final value of a specific metric, however, is not obtained simply by determining the average of a value such as speed – for each metric, a specific method is used to determine a value on a fixed range (so that all metrics can be used together).

Equation 5.1 shows the formal definition of a profile P as a set of metrics $m_1, m_2, ..., m_{nm} \in M$ with associated weights $(w_1, w_2, ..., w_{nm})$. Each profile may contain any number of metrics, which is equivalent to containing all metrics, some of them with a weight of zero. For simplicity reasons, the weight of each metric is defined in the range of zero to one, corresponding to a percentage of its importance in the final performance measurement. When calculating the performance value, the system converts these weights so that the sum of all weights is one (each weight is divided by the sum of all weights: $W_i = w_i / \sum_{j=1}^{nm} w_j$)

$$
\begin{aligned}
&M = \{m_1, m_2, \cdots, m_{nm}\} \\
&P = \{(m_1, w_1), (m_2, w_2), \cdots, (m_{nm}, w_{nm})\}, m_1, m_2, ..., m_{nm} \in M, w_1, w_2, ..., w_{nm} \in [0,1]
\end{aligned}
\tag{5.1}
$$

In order to combine the metrics being used in a profile, they must all produce a value on the same range, independently of any measuring unit or value being used by any specific metric. As an example, each vehicle has, in the respective configuration files, the definition of cruise speed and fuel consumption. Considering that the metrics are measured in the $[0,1]$ range, if a vehicle achieves an average speed during the simulation that is equal to its specified cruise speed, that *vehicle speed* metric will have a value of 0.5. As the average speed achieved by the vehicle during the simulation approaches its maximum speed, the metric approaches the value of 1. As the average speed decreases, so does the metric value, approaching 0 as the vehicle's speed converges to its stall speed, in the case of aircraft, or to zero, with other vehicles. Closely related to this metric, the fuel consumption metric has a similar and mirrored behavior – the metric value approaches 1 as the fuel consumption decreases. These two related metrics can be used together to form a third metric that relates speed and fuel consumption – this metric will measure the relation between the average speed achieved by the vehicle and the average fuel consumption. Similar metrics can be introduced relating two or more other metrics (for instance, a metric relating fuel consumption with the cargo transported by the vehicle, which can be useful when evaluating transport missions). As this tool evolves, the number of metrics available to the operator, both targeting individual vehicles and the entire team, will increase in number; also, the manner in which the final metric value is determined may be improved in some metrics, in order to promote a higher uniformity among metrics.

The Performance Analysis Tool allows for several simulation sessions to be combined, in order to obtain average values for several of the metrics. This allows for outliers to be dissolved, thus better approximating the actual average performance of the team. For instance, and considering as example a team with the mission of detecting a fire (with a random point of origin) in a given area, the performance of the team is dependent on the location of the fire – even if all vehicles repeat the same motion pattern in the several simulations, the fire can be detected early on or only later, depending on its location. By using several simulations, the values obtained by combining them together are better approximations for the average values of a given mission or task.

The tool can also allow for the comparison of two simulation sessions, or two sets of combined simulation sessions, using the previously calculated averages to compare them. This comparison can be very useful when comparing two variations of a given parameter. For instance, and returning to the example of a team searching for a fire, several search strategies can be compared by running several simulations for each search strategy and, after combining the results of the simulations with the same strategy, the performance values for the different search strategies can be compared.

This performance analysis can also be useful to the system, by acting as the input for a learning mechanism that can extract the best strategy or the best operational configuration for a given

mission under certain conditions.

This tool can also be used to synchronize the visual mission replay with an analysis of several parameters, such as the messages exchanged between the several agents during the simulation, or even the instant values and the evolution of several metrics throughout the simulation.

5.7 Summary

In this chapter, the main components of the developed platform were described in detail, as well as some of the interactions between components.

First, the simulator (considered a central component of the platform) was introduced in section 5.1, along with a description of the process that led to the choice of FSX (using a method that analyzes the simulators based on four main categories – simulation engine, graphics, fault injection and openness). Some of its possible applications and some adaptations that had to be made in order to use it with the developed platform were also presented.

Then, the Control Panel, the central component for interaction with the operator, was described in section 5.2. The Control Panel is used to configure all aspects regarding the simulation, including some general platform parameters, and also the configuration of scenario, team, disturbances and mission (according to the defined dialects).

After that, the Disturbances Manager was presented, describing its use as an alternative simulator for disturbances that cannot be simulated by or sensed through FSX. This component interacts with the Vehicle Control Agents, transmitting disturbance information that emulates sensor readings.

Then, the Monitoring Tools was presented, which is capable of monitoring the overall status of the simulation and also each individual vehicles's detailed status.

Finally, the Logging Tool and the Performance Analysis Tool were presented. The logging mechanism, which is actually performed by the several components of the platform, generates several log files, which can then be used by the Performance Analysis Tool to determine the performance of a team when conducting a mission, according to several metrics (either at a vehicle or team level).

For each of the described components, some improvements were identified and described at the end of the respective section.

The following chapter presents the remaining components – the ones with an autonomous behavior – and the chosen agent communications platform.

Chapter 6

Platform Agents

This chapter describes the components of the developed platform that present an autonomous behavior – ATC Agent and Vehicle Control Agents. Each of these applications is described and the interactions between components are specified. Also, some considerations regarding the Agent Communication Platform are made.

6.1 Agent Communication Platform

In order to facilitate communications among agents, a platform following FIPA guidelines was selected to be used. For that, the platforms presented in section 2.1.6 were analyzed and one platform was selected. Given that interaction with the simulation platform is performed using C♯, a .Net language, an initial selection process eliminated all Java-based platforms, which constitute the majority of the available platforms. This process resulted in three possible platforms targeting the .Net framework – CAPNET, ACENET and AgentService. The preferred use of Java as the implementation language for agent communication platforms could lead to the belief that it performs better. However, in [Hallenborg, 2008], the author compares one of the most popular agent platforms implemented in Java – JADE – with an implementation based on C♯, concluding that the performance of the platform increases when using C♯ specific constructs. This study shows that even though Java is the preferred language used in most open-source and agent communication platforms, it does not necessarily translate into a better performance.

Both CAPNET and ACENET seem to have been either abandoned or experiencing a very slow development, since no additional developments could be found on any of the projects since the initial publications (in addition to not finding any additional publication, the author was also unable to find a web site regarding any of the platforms). AgentService, on the other hand, registered a visible evolution over the years, with new features being added from time to time[1]. As such,

[1] No new developments, however, have been made since September 2009, when Andrea Passadore, one of the main developers, left the team. The project seems to be abandoned since then.

and given the functionalities and possibilities of AgentService, it was chosen as the platform for handling communications among the agent.

One important aspect regarding AgentService is its ability to support the connection of external applications, via the ExternalRuntime [LIDO, 2009a]. This feature provides the platform with the flexibility that allows for the integration of legacy applications and software that was not developed specifically for that platform. By using the ExternalRuntime, these applications are able to access all services provided by the platform (thus facilitating the reuse of any application developed beforehand). Another important supported feature is the ability to work in a federated environment [LIDO, 2009b]. This allows for multiple instances of AgentService to be running at different computers in a network, allowing for load balancing and also agent mobility (agents running within the platform can migrate to another platform within the federation, while maintaining their state, whereas programs using the external runtime need to disconnect from one platform and connect to another). A light version of the platform has also been developed, which can be executed in mobile devices, such as PDAs or SmartPhones, allowing agents to be executed in these devices [LIDO, 2008a]. The platform also provides support for the definition and use of ontologies [LIDO, 2007], as well as an Ontology agent [LIDO, 2008b] that provides ontology services, as defined by FIPA specifications [FIPA, 2001].

6.1.1 Services

In order to facilitate agent discovery, several classes of service types and names were devised.

For agents with only one instance in the platform, the service type *Service* is used. The name of the agent coincides with the agent function in the platform, as shown in Table 6.1.

Service Type	**Agent**	**Service Name**
Service	Control Panel	ControlPanel
	Monitoring Agent	MonitoringTool
	Logging Agent	LoggingTool
	Disturbances Manager	Disturbances

Table 6.1: Service Type and Name for Single-Instance Agents

For ATC Agents (Controllers) and Vehicle Control Agents, in addition to type and name, a subtype is also used: in the case of controllers, it indicates which role(s) it takes; in the case of a vehicle, it indicates the type of vehicle. In each of the cases, the name of the service corresponds to the unique identifier of the agent (in the case of an ATC Agent, the identifier starts with a *c* followed by a number; in the case of a Vehicle Control Agent, the identifier starts with an *a*, followed by a number). Table 6.2 summarizes these services. The service subtype uses a numbering that allows both controllers and vehicles to have different subtypes; for instance, an ATC Agent may have two roles simultaneously – Approach and Departure; a Vehicle Control Agent may also represent an amphibious vehicle (for instance, an amphibious aircraft can land on either land or water).

In addition to registering the names of the agents and this information, additional information is included in vehicle controlling agents, such as the sensors and cargo they carry, the team they

Service Type	Service Name	Service SubType
Controller	$c1$ $c2$ $\cdots$	1 - Approach 2 - Departure 4 - Ground 8 - Tower
Agent	$a1$ $a2$ $\cdots$	1 - Aircraft 2 - Land Vehicle 4 - Water Vehicle 8 - Underwater Vehicle

Table 6.2: Service Type and Name for Multiple-Instance Agents

belong to, and the identification of the vehicle within the simulated environment. They are registered with service type *Sensor*, *Cargo*, *Team* and *FSXID*, respectively. As the service name, the type of sensor, type of cargo, identifier of the team and identifier of the simulation object are used, respectively. Additionally, in the case of cargo, the subtype is used to indicate the amount being carried. The FSXID element is a very important one, since it establishes the relation between a simulated object and a team vehicle.

6.2 Air Traffic Control and ATC Agent

This section fully describes the ATC Agent, starting with a brief description of air traffic control operations, some existing software and automation approaches, and then detailing the implementation of the ATC Agent.

6.2.1 Air Traffic Control

Air Traffic Control (or ATC for short) encompasses a set of procedures, as well as people and technology, that assures secure operations of aircraft, both in flight and on land. The main goal of traffic control is to prevent collisions among aircraft or with other objects (stationary or mobile), while at the same time providing a fast and swift routing of aircraft throughout the terminal and providing pilots with information (such as weather conditions, for instance) [FAA, 2010a] [Sousa et al., 2010].

Collision prevention is accomplished through separation, vertical, lateral and longitudinal, which consists in maintaining a minimum distance between each pair of aircraft, in at least one axis.

Traffic control can be divided into two major areas:

- **Terminal**. Terminal control includes the control of aircraft (as well as other ground vehicles) on the airport surface and in the vicinities of the airport – usually a circular area centered on the airport, with a 30 to 50 nautical mile (56 to 93km) radius, from the surface to about 10.000ft (about 3.050m). Typically, terminal control is divided into several distinct control categories:

  - **Ground**. Ground Control is responsible for movements within the airport, which usually encompasses not only taxiways and inactive runways, but also other holding areas; it must ensure a fast and smooth operation of both aircraft and other ground vehicles within the airport.
  - **Tower**. Tower Control, or Local Control, is responsible for active runway management, clearing aircraft for takeoff and landing, and assuring that aircraft separation is maintained at all times; it must coordinate closely with Ground Control, to assure that aircraft moving in the airport can safely cross active runways, and that aircraft landing on the airport will not disrupt safe operations.
  - **Clearance**. Clearance Delivery is responsible for assigning aircraft with a route slot (considering both time and space constraints) after departure.
  - **Approach**. Also known as Terminal Control, it is responsible for handling traffic surrounding the airport, in a radius up to 50 miles, or 93km; it handles departures, arrivals and passing traffic, and must coordinate with the airport's Tower Control, other Approach Controllers, or en-route controllers.

  In some airports, some services (or even all of them) may be performed by the same controller; in others, smaller airports, one or more services may not be available.

- **En-route**. En-route control encompasses traffic between terminals. In order to facilitate control (especially in regions with a high traffic volume), airspace is often divided into several areas, each of which is assigned to a specific control center, responsible for managing aircraft entrance and exist from the area, as well as aircraft within the area.

ATC is a critical aspect of transportation nowadays, and ATC controllers are subject to a lot of stress [Myers, 2008]. As such, it has been a research subject for several years.

There are several existing software applications (some coupled with specific hardware) that are used for training purposes, or that can be used to make the work of traffic controllers easier[2] (refer to [Sousa, 2010] for more information on some systems).

One system that provides support for operations in an air traffic control tower is ACAMS[3]. This system, at work in several airports for over a decade, has a modular architecture that allows it to adapt to the idiosyncrasies of each airport, be it of military or civilian nature. Its flexibility allows for the operator to personalize the appearance of the application on the monitors, as well as the information to be shown and the manner in which it is presented. For fault tolerance reasons, the modules that compose the system communicate through a double Ethernet connection, working in a client-server architecture.

[2]Information on several systems can be found from `http://www.airport-technology.com/contractors/traffic/`

[3]More information available from `http://www.acams.net/`

Another system provided by NavCanada[4] [Crichton et al., 2001]. Its main system, EXCDS (extended computer display system), is an advanced tower, terminal, airport and en-route coordination system that allows controllers to manage electronic flight data. It allows for the integration with external information providers, thus extending its basic functionalities. It is in use several airports throughout Canada, and also in other countries. Another available product is SASS (Scheduling And Sequencing System), an arrival manager that assists in allocating landing slots and helps deal with traffic surges, thus allowing for a better airport efficiency.

One system that provides support for training is DATS (Durable Aviation Trainer Solutions), by BAE Systems C-ITS[5]. It uses commercially available off-the-shelf (COTS) hardware and works with standard operating systems, as to provide both affordability and ease of maintenance. DATS provides a comprehensive simulation of tower procedures, including a radar traffic simulator for large airports, local traffic simulation for airport management, and also pilot ATC interaction.

Several research projects have also been developed in this area, in an attempt to provide not only decision support functionalities for traffic controllers, but to make traffic control processes fully automated. Some works have tried to directly map the roles of ATC controllers under the current ATC organization into agents. One such example is described in [Callantine, 2002]. The authors extend CATS (Crew Activity Tracking System) [Callantine et al., 1999] to be used by several agents, each responsible for a sector of the airspace. The results were somewhat promising, since aircraft handover between sectors is performed with no problems, but the sectors closer to the airport tend to overload with traffic. In [Hexmoor & Heng, 2000], the authors use their previously developed ATC simulator, TACUND, to implement an ATC agent. This agent is aware of several information details of the aircraft in its airspace, such as location, direction, speed, desired altitude and intent to land. Through the use of three priority queues (one for landing requests; one for collision between aircraft; and one for aircraft in a holding pattern), it is able to avoid collisions and manage landings in the airport. This implementation provides the human controller with the ability to override the decisions made by the system, by means of a timer, thus providing a shared autonomy between agent and operator. The collision avoidance problem has also been tackled from a mathematical standpoint, as in [jen Chiang et al., 1997] - the authors provide a method for detecting collisions by analyzing the direction and speed of each aircraft and determining, one by one, virtual tubes that represent the course of the aircraft in a discretized space-time, which are considered obstacles the other aircraft cannot cross. Some other works implemented distributed collision avoidance mechanisms. One example is given in [Pěchouček et al., 2006], where each agent represents one aircraft. Each aircraft has four areas, each with a decreasing radius – communication, alert, security and collision. The communication and alert areas represent the range of communications with other aircraft and the collision radar range, respectively. The security area is an area around the aircraft where no other aircraft should be, for security reasons; if another aircraft is present in that area, situations such as turbulence are probable to arise. Finally, the collision area is used to determine whether a collision has occurred. The authors present a

[4]More information available from `http://www.navcanada.ca/`

[5]More information available at `http://www.baesystems.com/c-its`

cooperative protocol for avoiding collisions, and also a method to be used when there is no cooperation (which may be the result of a failure in the communications or collision radar systems). In [Krozel & Peters, 1997], the authors also use one agent to represent each aircraft. Each agent analyzes its own path for the near future, and if conflicts are detected, a decision is made (either using a greedy approach, or using a look-ahead strategy to detect possible conflicts that may arise after the first course adjustment). In [Gorodetsky et al., 2007], the authors use a mixed approach, with agents representing (or assisting) traffic controllers and agents representing (or assisting) pilots. The airspace is divided into one approach sector (the central area) and several arrival sectors (peripheral sectors). While in the peripheral sectors the pilot assisting agents are responsible for collision avoidance, in the central area, it is the responsibility of the controller assisting agent to ensure that all safety regulations are followed (and all aircraft must follow its directives). [Krozel et al., 2001] provides a comparison between centralized and decentralized approaches to maintain air traffic separation.

6.2.2 ATC Agent

The ATC Agent implements the concept of a centralized control for autonomous vehicles, and is meant to be used primarily for controlling traffic in limited areas, such as airports, ports or ground bases. Currently, the ATC Agent is responsible for handling traffic within an airport and in its vicinities [Sousa, 2010], according to controller definition – see section 4.2.6. However, this agent is expected to handle land and water traffic in ports and ground bases as well (a few minor adaptations are required to the current implementation). The flexibility of the ATC Agent in terms of configuration (see section 4.2.6 for more details) allows the aircraft to have different levels of autonomy when within a controlled area. This flexibility can be used to test the efficiency of centralized versus decentralized strategies in the control of air traffic in areas with a high volume of traffic, either in a continuous manner, or considering temporary traffic volume peaks. The ATC Agent can also be used for researching centralized coordination methodologies and techniques for air traffic management, as well as traffic management in water ports.

As mentioned in section 3.2.2, ATC agents are created by the Control Panel, when launching the scenario configuration (see section 4.2 for the scenario configuration specification). An instance of this agent exist for each controller defined in the Control Panel. When created, this agent automatically loads its configuration details, connects to both the simulator and agent communication platform, and starts monitoring vehicles within the area it has jurisdiction over. Figure 6.1 shows the configuration screen of the ATC Agent. Configurations can also be entered manually, for testing purposes, but are automatically obtained during normal operations.

Figure 6.2 shows the graphical monitoring screen of an ATC Agent, where the airport structure is visible, as well as the vehicles within the airport. On the right side of the screen, all messages sent and received by the ATC Agent are listed, showing message details. On the bottom of the screen, a list of all aircraft currently detected by the ATC Agent is shown, including information regarding the type of aircraft, its position, heading and speed, as well as an indication of whether

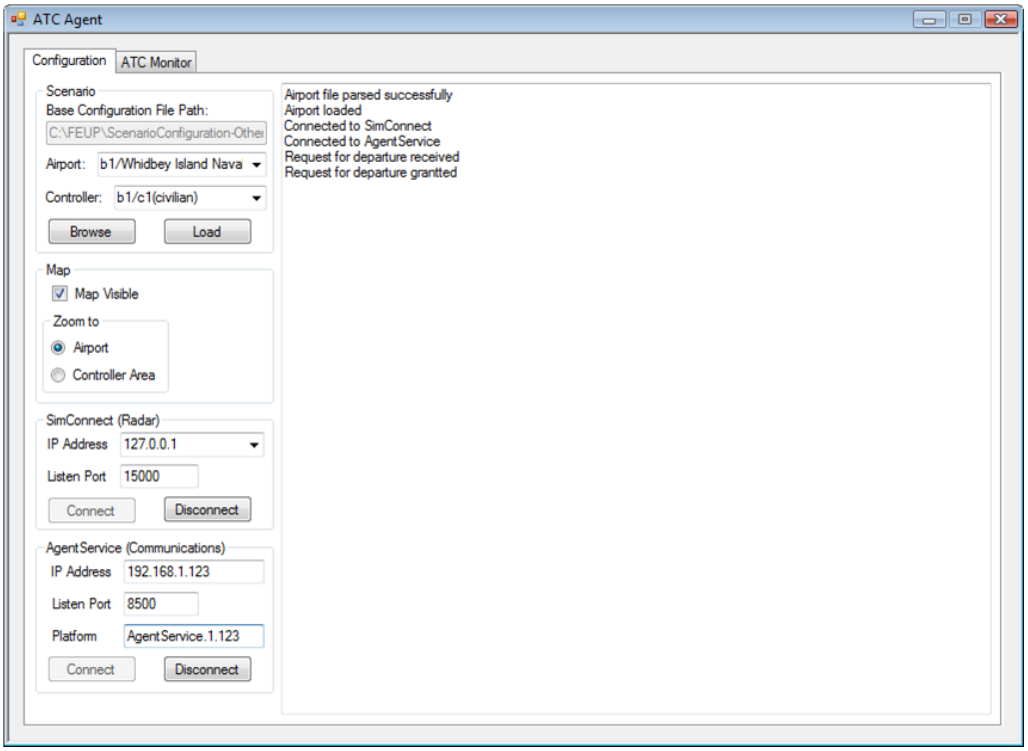


Figure 6.1: ATC Agent Configuration Screen

the aircraft is airborne or on the ground, and the state of the aircraft (see below for more information regarding the aircraft state).

6.2.2.1 Implementation Details

This agent needs to be aware of the physical layout of the airport in order to direct traffic in an effective manner. This is accomplished by using the description of the airport of the base of operations the ATC Agent is associated with.

As mentioned in section 4.2.2, taxiway elements have a specification that allows them to be easily interpreted as a graph. In order to avoid repetitive calculations during runtime, the shortest paths between any two points of the taxiway network are calculated during initialization and kept for future reference. Each taxiway was also divided into sections, each section being the straight connection between two consecutive points. This division not only facilitates graphical and path calculations, but also allows for a better representation of section occupancy and in the planning of the paths to be taken by the aircraft. Each occupied section will contain information regarding the aircraft that is currently in it, including its travel direction. This information can be used when more than one aircraft needs to travel in the same direction in the same taxiway – two aircraft cannot travel in the same taxiway in opposite directions, but they can travel in the same direction, if separation is maintained (which can be achieved, for instance, by allowing only one aircraft to be present in any given section at any time). Taxiways connect to runways, as well as to other elements, such as parking spaces or fuel facilities, which are represented using additional

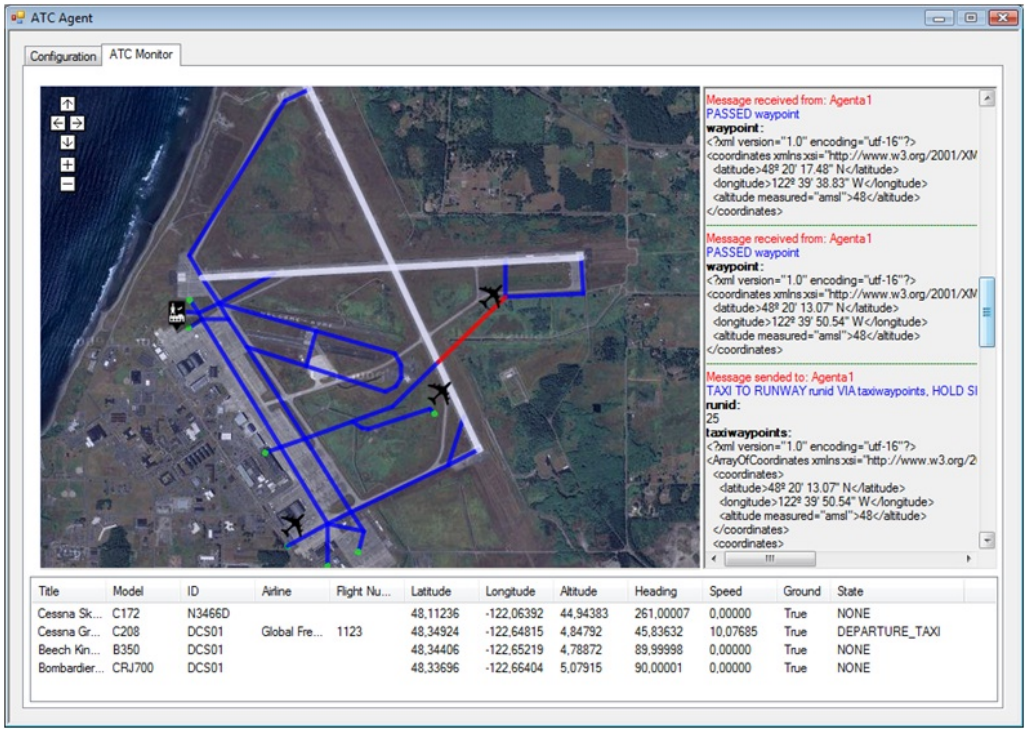


Figure 6.2: ATC Agent Monitoring Screen

connection points in the taxiway network. This representation allows for the ATC Agent to direct traffic not only from a parking space to the departing runway and from the landing runway to a parking space, but also to perform internal movements, such as directing an aircraft from a parking space to a fuel facility. Runways are kept in a list, containing information about interceptions with other runways (which are used when an aircraft is taking off or landing in one runway, to avoid conflicting traffic in the intercepting runway), as well as the connections with the taxiway network (this constitutes replicated information, but contributes to faster calculations in arrival situations, when the aircraft needs to vacate the runway as fast as possible as to allow other aircraft to land or takeoff at that runway).

Each aircraft within the controller area is attributed a state, according to their intended maneuver and the current state of interactions with the ATC Agent – Fig. 6.3 depicts these states and the possible transitions between states.

When an aircraft is first detected by the ATC Agent (either in the beginning of the simulation or when the aircraft enters the area under the agent's supervision), it is classified as being in the *None* state, since no intentions are yet known. From there, three actions are possible: the aircraft is airborne and requests to land in the airport; the aircraft is in the airport and wants to takeoff; the aircraft is in the airport and wants to move to another location within the airport. In the first case, the aircraft sends a message to the ATC Agent, requesting to land, which can be granted, if a runway is cleared of traffic, or delayed, by means of a holding pattern, if all runways are currently occupied. When the aircraft moves closer to the airport, it transitions to the *approach* state, where it should start to descend and align with the runway. After landing (transition to the *touchdown*

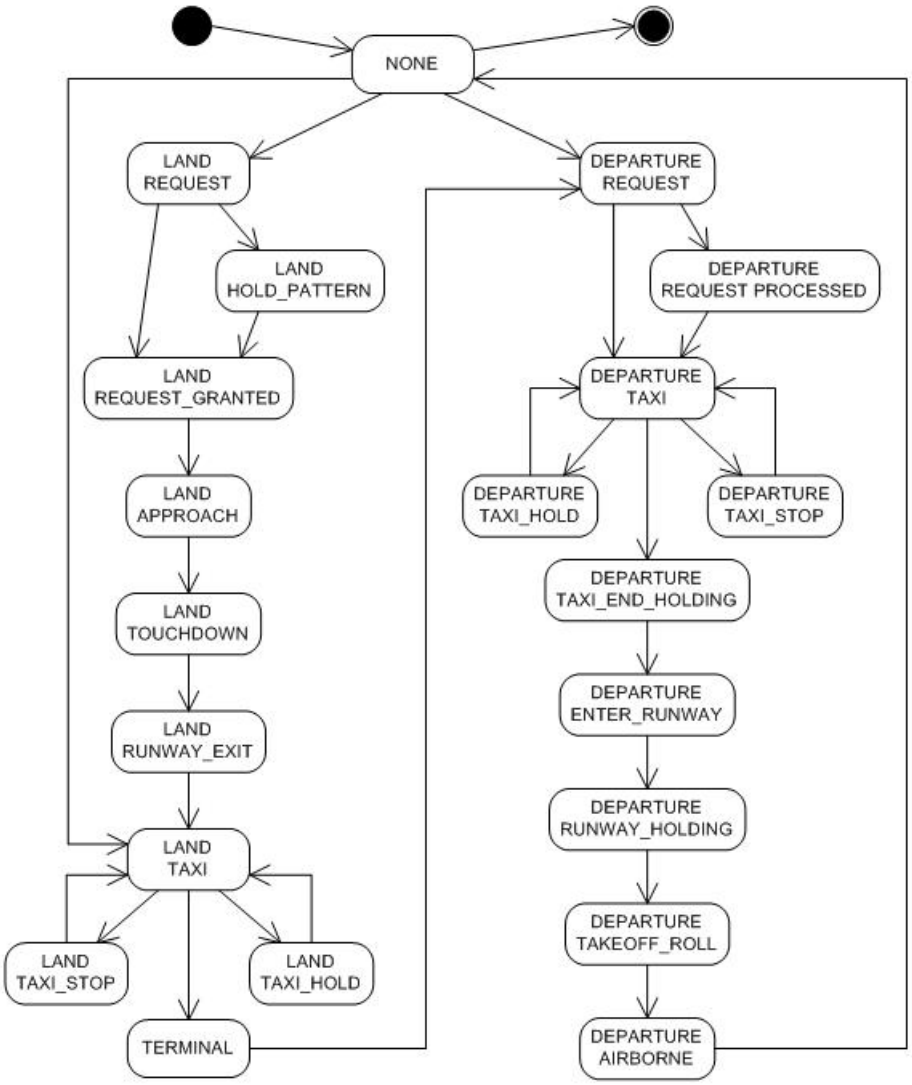


Figure 6.3: ATC Agent Aircraft States

state), the ATC Agent provides the aircraft with the taxi path for the desired destination (usually a parking space), transitioning to the *runway_exit* state, and the aircraft proceeds to taxi operations (described below), signaling its exit of the runway, so that other aircraft can use it (transition to the *taxi* state). In the second case, the aircraft sends a request for departure, which can be immediately granted, if the initial portion of the taxi path leading to the appropriate runway is clear, or delayed for a while (state *request processed*), until the aircraft can start moving. When the reply is sent to the aircraft, along with the taxi path for the departure runway (transition to the *departure_taxi* state), the aircraft performs the necessary taxi operations and comes to a stop before entering the runway, transitioning to the state *taxi_end_holding* (this safety measure not only ensures that no aircraft is currently using the runway, and also allows for a more efficient use of the runway). After receiving the order, the aircraft enters the runway (state *enter_runway*), aligns itself for takeoff and again stops (state *runway_holding*), waiting for the final clearance for takeoff – when the clearance

is received, the aircraft transitions to the *takeoff_roll* state and moves through the runway, taking off and transitioning to the *airborne* state. Finally, when the aircraft moves away from the airport, it again transitions to the *none* state. The third case, when the aircraft wants to move from one location to the other within the airport, can be considered as a particular case of either of the two previous cases, and consists only of the taxi operations.

Taxi Protocol The taxi protocol is used for directing traffic on ground, from one point of the airport to another. It is used as a sub-protocol for both takeoff (to direct the aircraft from the parking space to the departing runway) and land (to direct the aircraft from the runway to a parking space) protocols. As such, this protocol can be initiated either as part of the two mentioned protocols, or as a result of a request from a vehicle agent to an ATC Agent, to move from one point to another (for instance, a request to move from a parking space to a fuel facility, or a hangar). Figure 6.4 shows the taxi protocol interactions between the ATC Agent and the vehicle control agent.

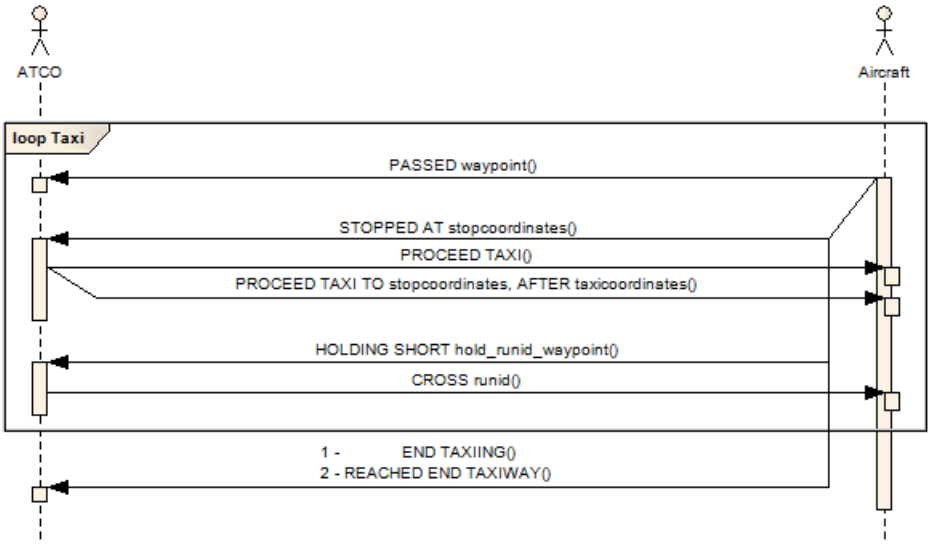


Figure 6.4: Taxi Protocol

The ATC Agent, after receiving the request, chooses a specific destination from all possible destinations (if more than one is possible, as can be the case of fuel facilities), and calculates the path the aircraft should take. This path can include crossing of one or more runways (in which case the aircraft must stop and wait for permission to cross the runway before doing so), and/or stop points within the taxiway network (these stop points are usually the result of intersecting taxiways, and, similarly to runway intersections, also result in the aircraft stopping and requesting permission to proceed). After receiving its route, the vehicle control agent calculates the appropriate speeds and sends the first leg of the route to the simulator – until the first stop point (taxiway intersection), or hold short point (runway intersection), if they exist; if not, the first leg is the only leg and consists of the entire route. As the vehicle moves, it communicates with the ATC Agent, informing it of its progress, and reporting each control point it passes through (using the *PASSED waypoint* message, as seen in Fig. 6.4). It also communicates being stopped at any hold short or

stop points (*HOLDING SHORT* or *STOPPED AT* messages, respectively). When clearance is received from the ATC agent to cross the runway, or to proceed taxi, respectively, the next leg of the route is calculated by the vehicle control agent and sent to the simulator. This process continues until the final destination is reached (Fig. 6.4 shows the two messages that can be sent to the ATC Agent when the taxi operation is embedded in a takeoff or land operation). The protocol also foresees the possibility of dynamically adding or removing stop points along the taxi path – this can be accomplished by means of the *PROCEED TAXI TO stopcoordinates AFTER taxicoordinates* message.

Takeoff Protocol The takeoff protocol is used when an aircraft wants to departure from the airport it is currently on. The vehicle control agent starts by requesting the ATC Agent for permission to departure to a given heading or specific location around the airport. The ATC Agent determines the best runway (and direction) for departure, as well as a taxiway route for the aircraft to reach the desired runway from its current location. By using the taxi protocol, as specified above, the aircraft reaches the desired runway and stops before entering the runway. When the runway is clear of traffic, the ATC agent instructs the vehicle agent to enter the runway and align itself for departure. After doing so, and communicating it to the ATC Agent, the clearance for takeoff is issued, and the vehicle agent accelerates and lifts off from the runway. The interactions used in the takeoff protocol are depicted in Fig. 6.5. These messages also mark the transitions of the aircraft state, as per Fig. 6.3, as explained above.

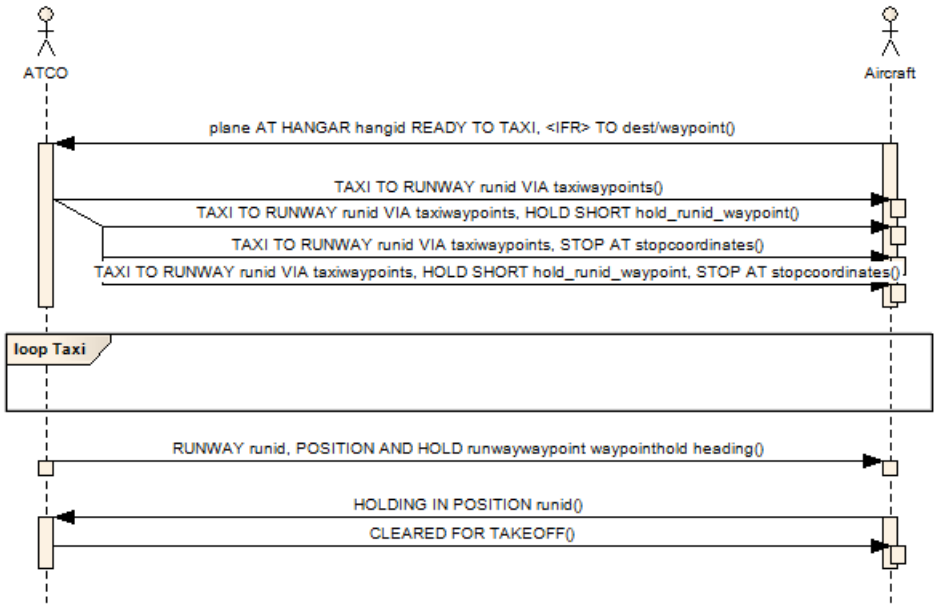


Figure 6.5: Takeoff Protocol

Land Protocol The land protocol is used when an aircraft is within the area controlled by the ATC agent and wants to land in the airport controlled by it. The vehicle control agents starts

by requesting the ATC agent for permission to land. The ATC agent first determines whether or not the aircraft can land (considering landing and takeoff operations currently under way). If the aircraft cannot land, the ATC agent instructs the aircraft to remain on a holding pattern near the airport, until permission to land can be given. When the aircraft is able to land, the ATC agent sends information regarding the approach to the airport and the runway used for landing. Using this information, the vehicle agent determines increasingly lower speeds for the approach, and sends the approach route to the simulator. Once the aircraft has successfully landed, the vehicle agent requests the ATC agent to taxi to a parking space, and proceeds using the taxi protocol, as specified above. Figure 6.6 shows the land protocol interactions between the vehicle agent and the ATC agent.

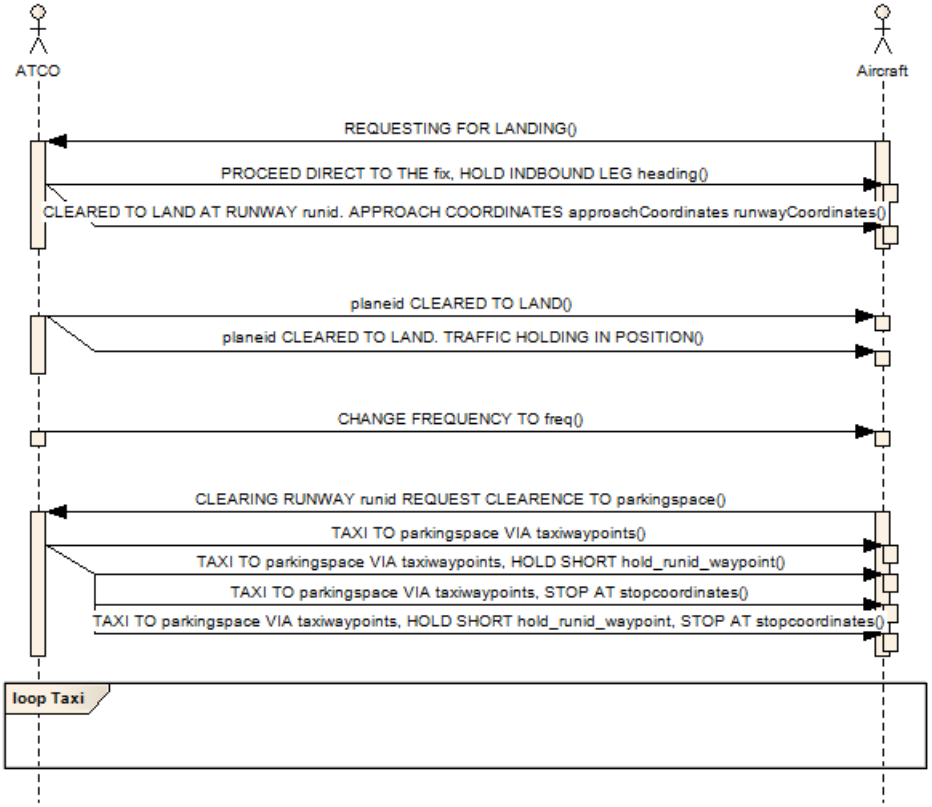
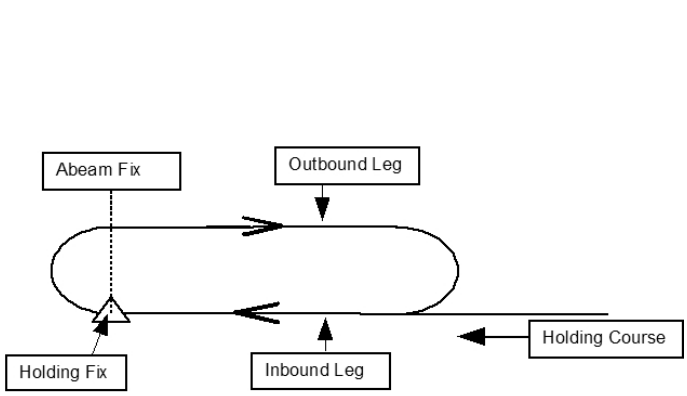


Figure 6.6: Land Protocol

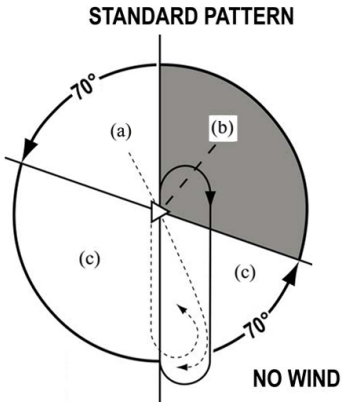
Holding Pattern The holding pattern is a maneuver performed at a constant altitude that consists of a series of right turns in a simple closed circuit, as can be seen in Fig. 6.7(a). The two elements necessary to define a holding pattern are the fixed point to which it reports (holding fix) and the heading of the pattern. The nature of the holding fix can vary, according to the aircraft or flight rules being used – it can be a visual reference (such as a bridge or a lake), a ground-based radio beacon (usually NDBs[6] or VORs[7] are used), or even a GPS point in space when GPS navigation is

[6]Non-Directional Beacon
[7]VHF Omnidirectional Range

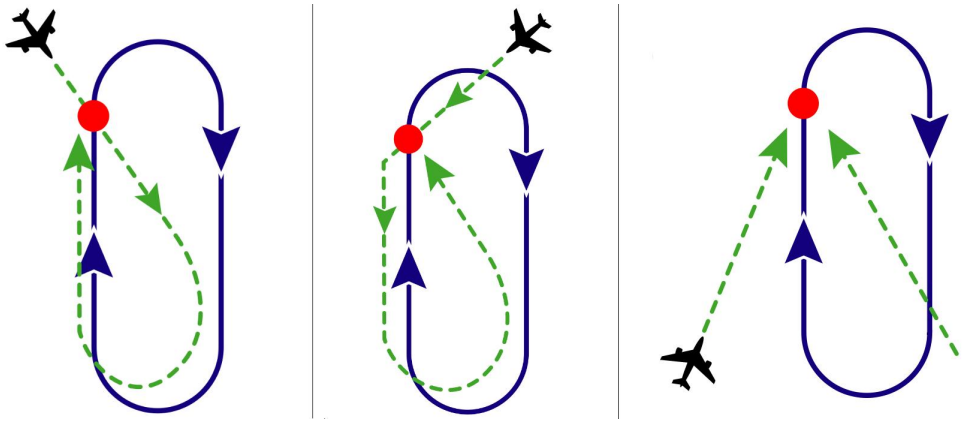
available. The length of the inbound/outbound legs is usually specified in time rather than length (one minute is the most common value used for the legs, resulting in four minute maneuvers). According to the angle in which the aircraft approaches the holding pattern, it will perform different maneuvers in order to enter the pattern. Figure 6.7(b) shows the three areas that result in different entries and Fig. 6.7(c) shows the entry maneuver for each entry area – for area a), the entry is called teardrop; for area b), the entry is called parallel; and for area c) the entry is designated as direct.



(a) Holding Pattern



(b) Holding Pattern Approach Angles



(c) Holding Pattern Entries

Figure 6.7: Holding Pattern Definition (a), Approach Angles (b) and Approaches

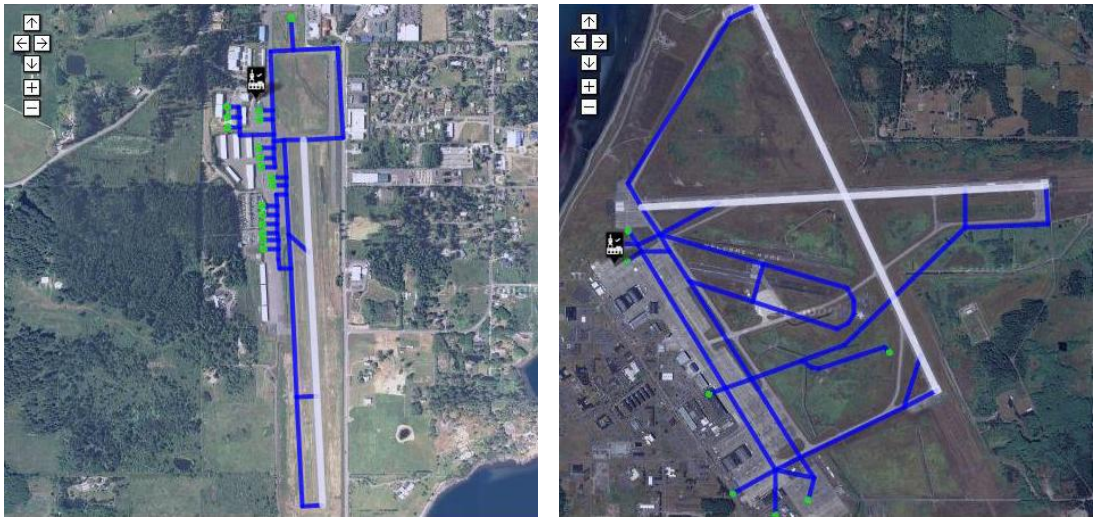
As to maximize space use regarding holding patterns, three holding patterns can be used, in a T shape – the main holding pattern is defined by a fix aligned with the runway and the same heading as the runway, and two additional fixes are defined on the left and right of the main fix (respecting the recommended 3 to 6 nautical mile distance between the different holding patterns), with their headings perpendicular to the runway heading. Also, if there is traffic requiring additional holding patterns, different altitude levels can be used, each of which sustaining the three described holding patterns.

6.2.3 Experimental Results

Some simple experiments were conducted as to validate the approach presented above. In order to test the impact of different airport layouts, two distinct airports were used:

- **Friday Harbor Airport**. This airport, shown in Fig. 6.8(a), has a simple configuration, with only one runway (in white), several parking spaces (green dots), and a simple taxiway network (in blue).
- **Whidbey Island Naval Air Station**. This airport, depicted in Fig. 6.8(b), contains two intersecting runways (in white), only a few parking spaces (green dots) and a more intricate taxiway network (in blue), with several intersection, thus providing a more challenging test scenario.

These two airports are located at a very close distance to one another (approximately $25Km$, both in the Northwestern region of the Washington state, in the United States. This distance is large enough to allow the ATC Agents to have non-overlapping areas, and small enough as to allow air traffic departing from one airport to arrive at the other one in a manageable time frame (which was deemed to be advantageous during the experiments).



(a) Friday Harbor Airport (b) Whidbey Island Naval Air Station

Figure 6.8: Friday Harbor Airport and Whidbey Island Naval Air Station

Several test were conducted in order to evaluate the performance of the ATC Agent:

- **Departures**. For this test, several aircraft were included in the team specification, all located at the airport being tested. All these aircraft issued a departure request within seconds of each other. In this case, the performance of the ATC Agent was measured by the average time between takeoffs (or the number of aircraft that takeoff during a given period of time).
- **Arrivals**. For this test, several aircraft were created at a different airport from the one being tested, and then commanded to fly in the direction of the area over which the ATC Agent

has jurisdiction, in order to issue a land request. In this case, the performance of the ATC Agent was measured by the average time between landings (or the number of aircraft that landed during a given period of time).

- **Departures and Arrivals**. For this test, the two situations considered above were recreated – several aircraft were created, both at the airport being tested and at the other one. The aircraft at the airport being tested issued a request to departure, and after takeoff would leave the ATC Agent area and immediately return, issuing a landing request this time. The aircraft originally at the second airport flew in the direction of the airport being tested and issued a landing request; after successfully landing and reaching a parking space, they issued a departure request. Using this setting, both operations (land and takeoff) were requested with similar frequencies, and both request types started at approximately the same time.

Regarding the departures test, the airport with a simple runway achieved an average time between takeoffs of approximately 82 seconds, which represents an average departure flow of forty-four aircraft per hour. In an additional note, the same values were achieved considering both takeoffs in the same direction and takeoffs in both directions of the runway. For the second airport, this value was considerably larger – the time between takeoffs was approximately 123 seconds (or little over two minutes), which corresponds to an average departure flow of 29 aircraft per hour. These results are explained by two facts: the aircraft in the second airport need to travel a greater distance to reach the runway before takeoff; both runways were used (50% of traffic for each runway), and since aircraft that used the runway shown as horizontal in Fig. 6.8(b) needed to cross the other runway to reach their departing position, there was additional delay in crossing the runway.

Regarding the arrivals test, both airports have a similar performance. For the first airport, the average time between arrivals is of 279 seconds, which results in an average arrival flow of approximately 13 aircraft per hour. For the second airport, the average time between landings is of 297 seconds, resulting in an average flow of approximately 12 aircraft per hour. The difference between results in the two airports is not significant, and can be explained by the larger runway length of the runways in the second airport, which results in an additional time for the aircraft to clear the runway, allowing the next aircraft to be granted permission to land.

The results of the third and final test are also similar for both airports. In both cases, the ATC Agent granted permissions for the departure and landing operations in an interleaved fashion, resulting in approximately double the number of operations, when compared to the second test (arrivals). The time between operations varies significantly, since a takeoff is performed in a relatively expedite manner after a landing, but the aircraft landing after a takeoff operation takes more time to reach the airport. For the first airport, an average of 25 operations per hour is achieved, while on the second airport that average is of 24 operations per hour.

Figure 6.9 shows a graphical representation of these results, comparing the number of departures, arrivals and the total number of operations per airport per hour.

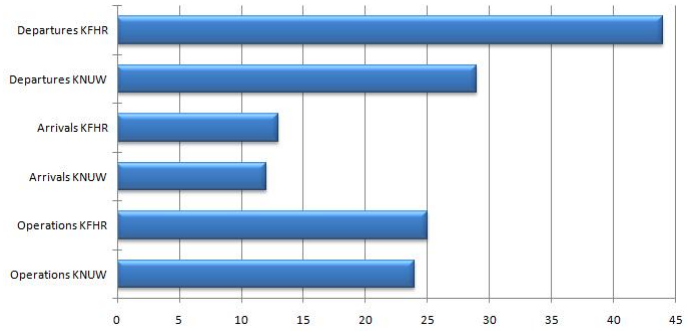


Figure 6.9: Comparison of Departures, Arrivals and Total Number of Operations per Airport

The results of the conducted experiments show that the approach is valid, even though some improvements can be made to the implementation, as described in the next section.

6.2.4 Summary

In this section, air traffic control operations were introduced, as to provide with the necessary background to understand both the research area in which this agent can be used, and also the operations that it needs to handle. Then, the ATC Agent was described in some detail, including the specification of some implementation details, namely the management of the aircraft within the controller area, and the interactions with the Vehicle Control Agent (which is described in more detail in the following section). The conducted experiments show that even though the current implementation could be improved, it proves the feasibility of a central agent to direct traffic in areas with a high traffic density.

Some lines of work have already been identified that would improve the performance of the ATC Agent:

- When multiple aircraft are traveling along the same taxiway on the same direction, separations is maintained by allowing only one aircraft to be present at any given section at a time. This solution, though simple, has one drawback – in taxiways with long sections, such as the ones in the Whidbey Island Naval Air Station (see Fig. 6.8(b)), the distance between aircraft is unnecessarily large. This problem can be solved using two distinct approaches, both allowing for a (theoretically) improved performance on ground operations:

  - Using information regarding the aircraft size and location, as well as the taxiway network configuration to determine a point to be used as a virtual intersection on the taxiway section. This approach is supported by the *PROCEED TAXI TO stopcoordinates AFTER taxicoordinates* message, as seen in the taxi protocol, but requires calculations to be performed during runtime.

  - Breaking down long and straight taxiway sections into several sections. This process can be done automatically by the ATC Agent during the initialization process and

does not require any modifications to the existing protocols, or any calculations to be performed during runtime.

- Performance in landing operations can also be improved. Two measures can contribute to this improvement:
  - Currently, when a runway is assigned to an aircraft for landing, it becomes unusable until the aircraft lands and vacates the runway. However, from the time the runway is assigned to the moment the aircraft actually lands, there is ample time to allow for aircraft to cross the runway, or even to takeoff from the landing runway (if takeoff is in the same direction as landing) or a crossing runway.
  - Currently, an aircraft is given permission to land only when a runway is clear and no other aircraft is attempting to land on the runway or any intersecting runways. However, due to the time hiatus between the moment landing permission is given and the moment the aircraft actually lands, several aircraft could be granted permission to land, provided they have the necessary spacial and temporal separation, so that one aircraft has time to land and clear the runway before the next one lands.
- Related to the previous item, the ATC Agent still cannot use runways to their full potential when controlling an airport with two or more intersecting runways. Several measures can be taken to improve this situation, such as the ones considered above for improving performance in landing operations.

The following section describes the component of the platform that the ATC Agent interacts with the most – the Vehicle Control Agent.

6.3 Vehicle Control Agent

Each instance of the Vehicle Control Agent controls one vehicle. When created by the Control Panel, this agent automatically loads its configuration details, based on the specific vehicle it represents (see section 4.3.1) and the vehicle type definition (see section 4.2.7). It then connects to both the simulator and agent communication platform and creates the vehicle within the simulator, expecting for a mission to be defined in the Control Panel and loaded. If the vehicle is within an area under the supervision of a traffic controller, it establishes communication with it, according to the level of autonomy defined for the controller.

The vehicle control agent is internally organized in several modules, as can be seen in Fig. 6.10, each of which can be further sub-divided according to functionalities (some of these modules are analyzed in more detail in the following sections). The SimConnect module handles all communications with the simulator (and eventual real vehicle it is associated with), and provides an interface for the agent to interact with the simulator. Two modules can be found within this one – the submarine navigation module (described in more detail below) and the real vehicle module, which allows for the communication with the actual vehicle the Vehicle Control Agent represents.

The agent interaction module handles all communications with other agents within the platform. Several modules can be identified within this module, as to separate communication with distinct agents (the most important ones, the communication with an ATC Agent, communication with other Vehicle Control Agents, and with the Disturbances Manager are depicted in the figure). The vehicle maneuvering control module is responsible for translating high-level maneuvers or motion intentions, which are usually generated by the planning and high-level reasoning module, into low-level maneuvers that can be interpreted by the SimConnect module as interactions with both simulator and real vehicle. Finally, the vehicle monitoring module is responsible for self-assessment, continually monitoring the vehicle in order to detect possible failures – if one is detected, appropriate measures need to be taken in order to ensure the security of operations.

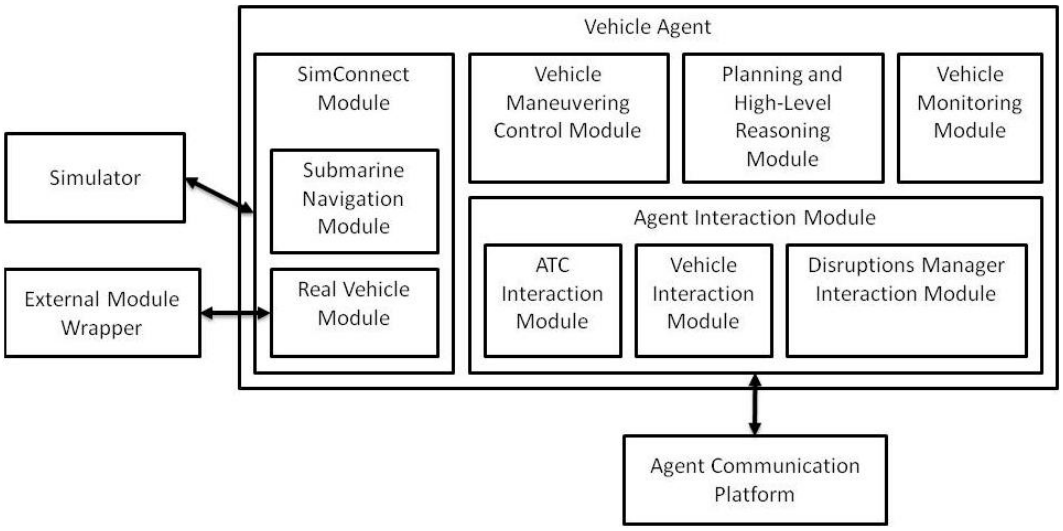


Figure 6.10: Vehicle Agent Internal Architecture

Figure 6.11 shows the interface of the Vehicle Control Agent used for development and testing purposes. The first tab (*Connection Setting*) allows for the manual configuration of the connection to Flight Simulator X, including the choice of the vehicle to be loaded, as well as its location and orientation. During normal operations, these details are obtained from the information introduced in the Control Panel for both scenario and teams, and sent to the Vehicle Control Agent when the team is launched.

The second tab, which is visible in Fig. 6.11, shows a high-level control override interface that allows the operator to send several maneuvers to the vehicle. These maneuvers are presented in section 6.3.2, along with a brief description on how the these maneuvers are synchronized with the simulator.

Both the *Aircraft Monitoring* and *Logging* tabs are shown in Fig. 6.14(a) and 6.14(b) below. The *Aircraft Monitoring* tab allows the operator to monitor the position of the vehicle using a Google Maps plugin (this implementation served as a base to the monitoring tool shown in section 5.4.1). Monitoring the individual vehicle within this tab needs to be started manually and can also be stopped, as to reduce the workload of the computer running the Vehicle Control Agent. The

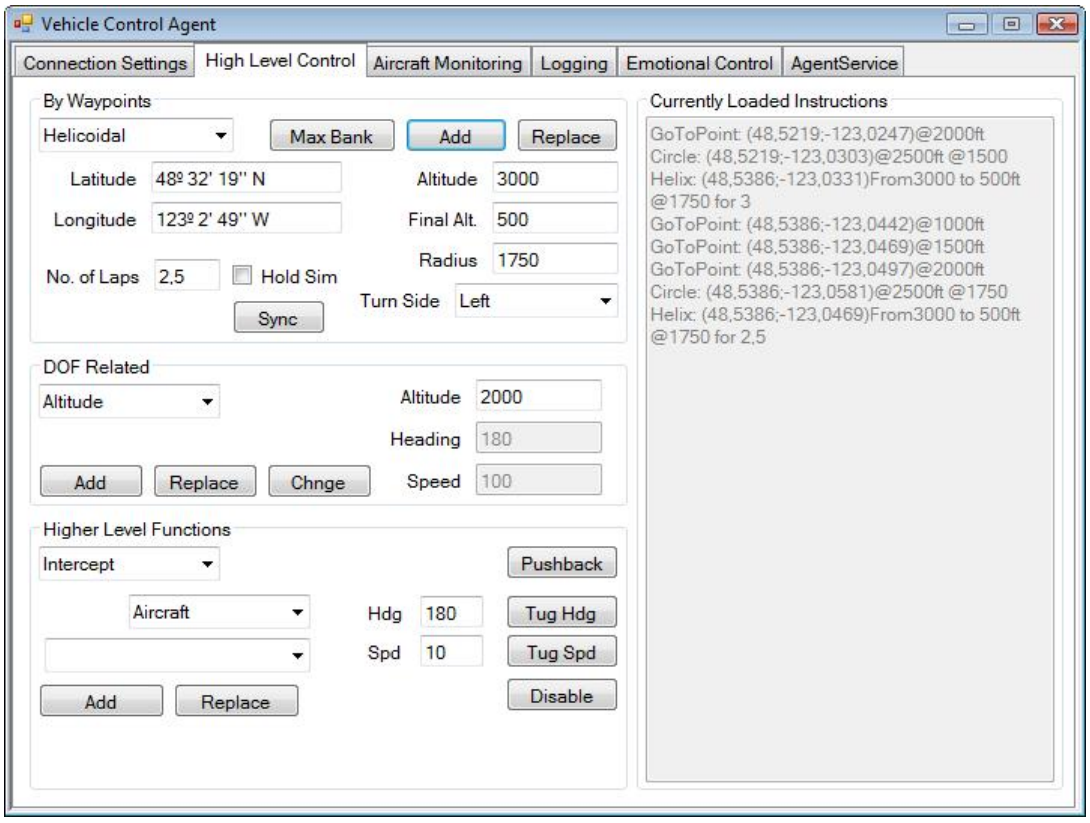


Figure 6.11: Vehicle Control Agent Interface

Logging tab allows the operator to manipulate individual log files and sessions (see section 6.3.1.6 for more information on these features).

The *Emotional Control* tab was used to interact with the emotion classifier module, as presented in section 5.1.2.2. Finally, the *AgentService* tab provides with a means to monitor the connection to the agent communications platform, and presents a list of all messages sent and received by the Vehicle Control Agent, showing message details such as sender, receiver and content of the message, in a manner similar to what was presented for the ATC Agent, in section 6.2.2, and seen in Fig. 6.2, on the right panel.

6.3.1 Vehicle Control

One of the most important aspects to consider after the choice of simulator has been made is how to control a vehicle within the simulator. After some consideration of the FSX interaction possibilities and simulation engine capabilities, three control strategies were deemed possible – external (direct manipulation of the aircraft control surfaces, such as aileron and elevators through a PID controller), aircraft's autopilot system (mimic the actions of a pilot handling the autopilot system) and simulator generic AI autopilot system. In order to determine the best approach, all three strategies were implemented and tested, using different aircrafts, in distinct weather conditions, and using a series of maneuvers with increased degree of complexity. The performance

of each strategy was analyzed according to maneuver effectiveness and communication requirements (with the simulator), based on the experimental results [Silva et al., 2009a]. The following sections detail the three vehicle control approaches, how basic maneuvers are computed and executed, the experimental settings and the results of the experiments and finally the conclusions, supporting the chosen strategy.

6.3.1.1 PID Controller

The first approach to controlling an aircraft within the simulation environment is to recreate the actions of a pilot when attempting to follow a certain route. This was accomplished by direct manipulation of the main controls of the aircraft, namely the throttle, aileron and elevator. The throttle control is used to control the speed of the aircraft. The elevator control is used to control the pitch angle of the aircraft (sometimes referred to as angle of attack, pitch measures the nose up or down angle of an aircraft), and, when used in conjunction with the throttle control, it allows to control the altitude of the aircraft – in order to climb, the elevator must be set to a positive value, and the throttle should be set to full throttle; in order to descend, the elevator must be set to a negative value, and the throttle should be set to idle. The aileron control is used to control the bank angle of the aircraft (also called roll, it is the angle of rotation of the plane about its longitudinal axis). In order to make a turn, the aircraft rolls toward the inner part of the desired turn. By using these controls together, higher-level maneuvers are executed.

In order to handle these aircraft controls, a PID (Proportional-Integral-Derivative) controller was used. A PID controller is a generic control loop feedback mechanism that attempts to correct the error between a measured variable and the desired value by calculating a corrective action that can adjust the process accordingly. It is a well-known controller from the control theory field, and has been used successfully in the industry for many years. Its numeric implementation consists on the evaluation of Eq. 6.1, where e is the difference between the reference value and the feedback measured value, τ is the time in the past contributing to the integral response and K_p, K_i and K_d are the proportional, integral and derivative gains, respectively. The proportional term adjusts the output signal in direct proportion to the error, the integral term is proportional to both magnitude and duration of the error, and the derivative term measures the approximate rate of change of the error. Tuning the gain factors of the PID controller is an important step, to assure optimal values for the desired control response [Skogestad, 2003].

$$output(t) = k_p e(t) + k_i \int_0^t e(\tau)\, d\tau + k_d \frac{de}{dt} \quad . \tag{6.1}$$

For this approach, the control agent communicates with the simulated aircraft at a given rate, in order to send the current values, calculated with the PID controller. Ideally, the communication would be uninterrupted, since continuous adjustments to the aircraft controls should be made in order to maintain the desired flights settings. However, the simulator only sends and accepts values once per simulation cycle, and to send data at a higher rate would be a waste of processing time. Moreover, one has to account for communication latencies that make it difficult to receive data

from a given simulation cycle and to send the corrective values that reach the simulator in time for the following cycle.

There is a simulation variable that is modified by the control agent for each of the three aircraft controls that are manipulated by the controller, and other variables that can be read in order to assess the current position, speed and attitude of the aircraft, used in the calculations as mentioned above.

In order to make a banked turn with a given radius (for circling maneuvers, for instance), some additional information has to be taken into account. The radius of a banked turn is given by Eq. 6.2, where v is the true airspeed of the aircraft, g is the acceleration due to gravity and θ is the bank angle of the aircraft.

$$R = \frac{v^2}{g \tan \theta} \quad . \tag{6.2}$$

Given Eq. 6.2, one can determine the necessary bank angle to generate a banked turn with a given radius at a given speed.

6.3.1.2 Autopilot

In this approach, the autopilot features of aircraft are used. Autopilot systems assume various forms, ranging from simple wing-levelers to complete systems, capable of controlling an aircraft from takeoff to land. Most modern aircraft feature altitude, heading and speed controls, allowing the pilot to set the desired values for each feature. The altitude control manipulates the elevator and throttle (if auto-throttle is available) in order to reach and maintain a certain altitude. The heading control manipulates the aileron in order to make a banked turn until the desired heading is reached, and the speed control automatically adjusts the throttle to maintain the desired airspeed. This approach uses the available autopilot systems as a pilot would, to adjust the aircraft's course to the desired settings.

There is a straightforward limitation to this approach, which is the necessity for the presence of an autopilot system. In fact, many smaller, older aircraft do not possess an autopilot system, or it is only comprised of the already mentioned wing-levelers, capable only of maintaining a straight level flight (but unable to change altitude, heading or speed). The authors, however, feel that this limitation does not impose a major problem, since there are several commercially available autopilot systems for small unmanned aircraft, such as the Piccolo autopilot system, from CloudCap Technology [CloudCap Technology, 2008], which are usually used for the type of aircraft that are intended to be used in real-life experiments.

With this approach, there are limits that need to be taken into account when planning the maneuvering scenarios, such as the autopilot system limits for the bank angle, for instance. As previously seen, the bank of a plane influences the radius of the curve, and as such, a minimum radius for turns is calculated based on the aircraft cruise speed and maximum bank angle at the beginning of the simulation, and that minimum value is enforced on maneuvers that require the plane to move in a circular fashion.

Since these systems do not allow for a direct control of the bank angle (some systems have the capability to further limit the maximum bank angle to usually half of that value, to produce smoother flights), a circular path is a bit more complicated to achieve. For circular paths, several points along the turn are calculated, and used as a basis for heading determination. By providing the aircraft with regular changes in the desired heading, a smooth circular path is achieved. The number of points in the path is directly proportional to the difference between the desired radius and the minimum radius – if the desired radius coincides with the minimum radius at the current aircraft speed, there is no need to use more than two alternating points to update the heading control. As with the PID approach, this approach also requires frequent communication with the aircraft, in order to send the desired values for the autopilot system.

6.3.1.3 AI Autopilot

The third approach makes use of the AI autopilot within the simulator to control the aircraft. This AI autopilot can be used in every aircraft, independently of the existence of an autopilot system such as the one described in the previous approach. It is used by the simulator to guide the generated air traffic from departure to arrival airports.

As with the previous approach, this autopilot also features limitations, namely the bank angle. When an autopilot system is available, the maximum bank angle is used to calculate the minimum radius for a turn, just as in the previous approach. When no autopilot system is present, the default value of twenty-five degrees is used – this is the most common value for autopilot systems, and experimental activities have demonstrated that this is also the most usual value for this AI autopilot.

This approach has, however, a few more limitations. In this case, all navigation is based on waypoints, which specify not only the latitude/longitude/altitude coordinates of the desired point, but also the desired speed or throttle percentage to be applied. Having this limitation in mind, and similarly to the previous approach, several points have to be calculated and fed to the aircraft for circular paths. This approach, however, does not require frequent communication with the aircraft – all points of a maneuvering sequence can be sent at the same time, in a waypoint list structure. The exception to this behavior is the existence of loops in the path – if a portion of a path is to be repeated until some external event occurs (such as human intervention), the waypoints that represent that loop have to be sent at the beginning of that loop, with a flag indicating that after the last waypoint, it should return to the first; when the loop is to be broken, the remaining waypoint must then be sent, replacing the previous ones.

6.3.1.4 High-Level Maneuvering and Waypoint Computation

In order to compare these methodologies, some high-level maneuvers were considered, namely the 'go to point' instruction, the 'circle' and the 'helix' maneuvers. Additional instructions could be issued to the aircraft, including desired airspeed, heading or altitude and target vehicle interception. These additional instructions would force the aircraft to accelerate or decelerate to attain

the desired airspeed, turn and maintain the desired heading or climb or descend in order to reach and maintain the given altitude. The target interception instruction provides the aircraft with the most probable point of interception with the target vehicle, if possible (aircraft and target vehicle speed, distance and heading are used to determine if interception is possible), updating the interception point every few seconds. The three first instructions are sufficient to assess the efficiency of vehicle control for the purposes of this test, since other maneuvers can be achieved by using a combination of the first ones or a similar method – for instance, changes in altitude can be achieve by using two 'go to point' instructions, changes in heading can be achieved by using a 'circle' maneuver with a fractional number of laps, and even takeoff and landing operations can be achieved resorting to several 'go to point' instructions.

The 'go to point' maneuver has three obvious parameters: latitude, longitude and altitude of the point to be reached. The altitude is perceived as altitude above the mean sea level (and not altitude above ground level). The circle maneuver, also called loiter, is defined by the central point (latitude, longitude and altitude), the radius of the circle and the number of laps, if not to loop indefinitely. The helix is defined by a central line (latitude and longitude), the initial and final altitudes, the radius of the helix and the number of laps. As previously mentioned, the radius for the last two maneuvers is conditioned by the aircraft's maximum bank angle and airspeed. Also, the number of laps to be executed in the helix maneuver is constrained by the difference between initial and final altitudes and the aircraft's maximum safe rate of climb/descend (vertical speed). Algorithm 3 and 4 illustrate how the circle and helix maneuvers are computed.

Algorithm 3 Circle Maneuver Algorithm

$CircularListpoints \leftarrow generatePoints(center, radius)$
if $AddingManeuver$ **then**
  $position \leftarrow getLastPoint()$
else
  $position \leftarrow getCurrentPosition()$
end if
$waypoints \leftarrow orderPoints(center, points, position, turnSide)$
if $nTurns \leftarrow 0$ **then**
  $waypoints[waypoints.Length - s1].Flag \leftarrow WRAP\_TO\_FIRST$
  return $waypoints$
else
  $wps.Length \leftarrow waypoints.Length \times nTurns$
  for $i = 0$ to $wps.Length - 1$ **do**
    $wps[i] \leftarrow waypoints[i \bmod waypoints.Length]$
  end for
  return wps
end if

In order to compute the points used by the latter two maneuvers in the two approaches that use autopilot systems, some considerations were made. First, the number of points to consider is directly proportional to the difference between the intended radius and the aircraft minimum turn radius – Alg. 5 illustrates how these points are determined. Second, and most importantly,

Algorithm 4 Helix Maneuver Algorithm

```
CircularList points ← generatePoints(center, radius)
if AddingManeuver then
    position ← getLastPoint()
else
    position ← getCurrentPosition()
end if
waypoints ← orderPoints(center, points, position, turnSide)
wps.Length ← waypoints.Length × nTurns
for i = 0 to wps.Length − 1 do
    wps[i] ← waypoints[i mod waypoints.Length]
end for
step ← (finalAltitude − initialAltitude)/(wps.Length − 1)
for i = 0 to wps.Length − 1 do
    wps[i].altitude ← initialAltitude + step × i
end for
return wps
```

the order in which those points are presented can deeply influence the path of the aircraft. For a smoother transition, a tangential approach to the virtual circle is preferred, especially when the circle has a radius closer to the minimum turn radius. The following algorithm was used to determine the first point and the order of the remaining points: a list of points is determined for the given radius, ordered as to form a circle; the distances between the aircraft and the points are determined; the first point is the $(N/2)^{th}$ closest point to the aircraft, N being the number of points in the circle; the second point is the point further away from the aircraft, chosen among the two points that are adjoining the first point in the original list of points. The remaining points are determined according to the order in which they were initially determined, as shown by Alg. 6. Figure 6.12(a) illustrates the choice of the first and second points in a hypothetical 8-point circle and subsequent direction of turn.

Algorithm 5 Generate Circular Points Function

```
nPoints ← determineNPoints(radius, vehicle.getMinTurnRadius())
headingStep ← 360/nPoints
for i = 0 to nPoints − 1 do
    points[i] ← addDistanceHeading(center, radius, headingStep × i)
end for
return points
```

The current aircraft position is used if the circle or helix maneuver is to be executed immediately. If the maneuver is to be queued after other maneuvers, the final point of the previous maneuver is used. Also, the aircraft position to consider may be adjusted to a position where the aircraft's heading is towards the region of the desired maneuver, considering a turn with the minimum radius at current speed, as illustrated in Fig. 6.12(b).

Algorithm 6 Order Circular Points Function

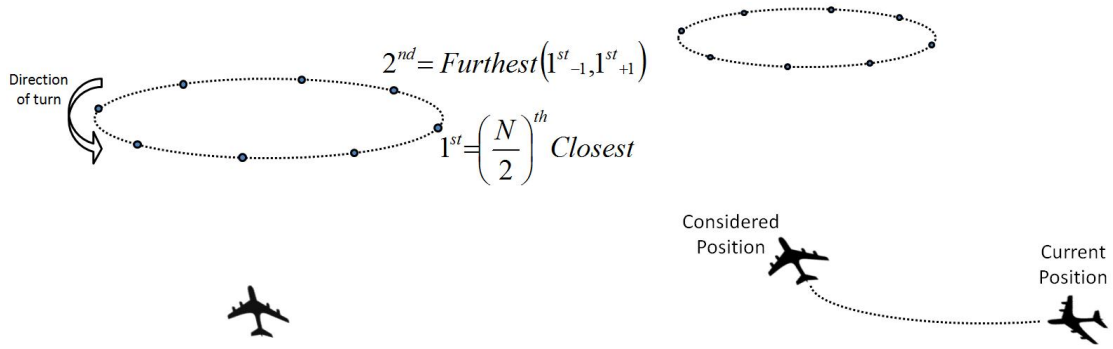
```
distances.Length ← points.Length
for i = 0 to points.Length − 1 do
    distances[i] ← distance(position, points[i])
end for
distances.Sort()
firstPoint ← points.byDistance(distances[points.Length/2])
distanceOne ← distance(position, points[firstPoint + 1])
distanceTwo ← distance(position, points[firstPoint − 1])
secondPoint ← points.byDistance(max(distanceOne, distanceTwo))
waypoints.Length ← points.Length
for i = 0 to waypoints.Length − 1 do
    if distanceOne ≥ distanceTwo then
        waypoints[i] ← points((firstPoint + i) mod points.Length)
    else
        waypoints[i] ← points((firstPoint − i) mod points.Length)
    end if
end for
return waypoints
```

6.3.1.5 Experimental Settings

Different experimental conditions were recreated for each control approach, with three main controllable variables – aircraft type, weather conditions and maneuver sequence.

Different aircraft types were used to assess how generic the approach was, or if it had limitations, namely regarding the size or type of aircraft. A total of three distinct aircraft were used in the experiments:

- **Piper J-3 Cub**. A single-engine two-seater light aircraft, with a wingspan of 10.6 meters and a maximum speed of 74 knots – Fig. 6.13(a);



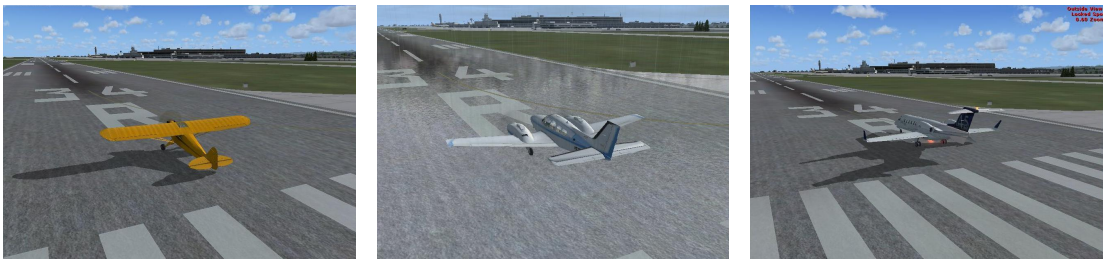
(a) Waypoint Determination (b) Aircraft Position Adjustment

Figure 6.12: Waypoint Determination and Aircraft Position Adjustment

- **Beechcraft Baron 58**. A twin-engine six-seater with a wingspan of 11.5 meters and a cruise speed of 200 knots – Fig. 6.13(b);
- **Bombardier Learjet 45**. A twin-engine-jet nine-seater with a wingspan of 14.6 meters and a cruise speed of 464 knots – Fig. 6.13(c).

Since this project is to be used with small to medium-sized aircraft, there was no need to conduct the experiments with larger or faster aircraft. Although the Piper J-3 Cub does not possess an autopilot system, it was important to include this aircraft, as to test the performance of the two other approaches on a smaller, slower plane.

Different weather conditions were used to test the control performance under adverse conditions, and to see how flexible the approach was to the existence of uncontrollable and unpredictable external factors. The experiments were conducted under two different weather conditions: 'fair weather' (shown in Fig. 6.13(a) and 6.13(c)) and 'gray and rainy' (shown in Fig. 6.13(b)). Harsher weather conditions were not tested, since the intended aircraft do not usually fly under more adverse weather conditions.



(a) Piper J-3 Cub (b) Beechcraft Baron 58 (c) Bombardier Learjet 45

Figure 6.13: Piper J-3 Cub, Beechcraft Baron 58 and Bombardier Learjet 45

Different high-level maneuvering sequences were also tested, as to assess how the approach would handle maneuver transition and how smooth the final flight would be. Basic high-level components were used to compose the three maneuvering scenarios used in the experiments. These high-level instructions include a 'go to point' command, indicating that the aircraft should pass through a given latitude/longitude/altitude point; circle, indicating that the aircraft should circle a given point with a given radius, either one time or continuously; helix motion, either climbing or descending from one initial altitude to a final altitude though a series of turns around a central point.

All experiments begin with the selected aircraft stopped in the end of a runway (34R) of the selected airport – Seattle-Tacoma International – with idled engines, facing the runway (see Fig. 6.13). The first command of the maneuvering sequence is always a 'go to point' command, with a point directly in front of the aircraft, approximately 500 feet above the runway, in order to assure both a smooth takeoff and that the aircraft attains its cruise speed. Slight variations in the scenarios were introduced for the three aircraft, due to their different cruising speeds. As already mentioned in the section above, the radius of a banked turn is proportional to the square of the

velocity of the aircraft, and hence, a larger radius was used for faster aircraft in the circle and helix commands. As a result, other commands were also modified in terms of longitude/latitude, in order to accommodate for the larger radius of these commands, in an attempt to maintain the overall proportions between the several control points.

Data from the simulation sessions was collected, as to be further analyzed, as described in the following section.

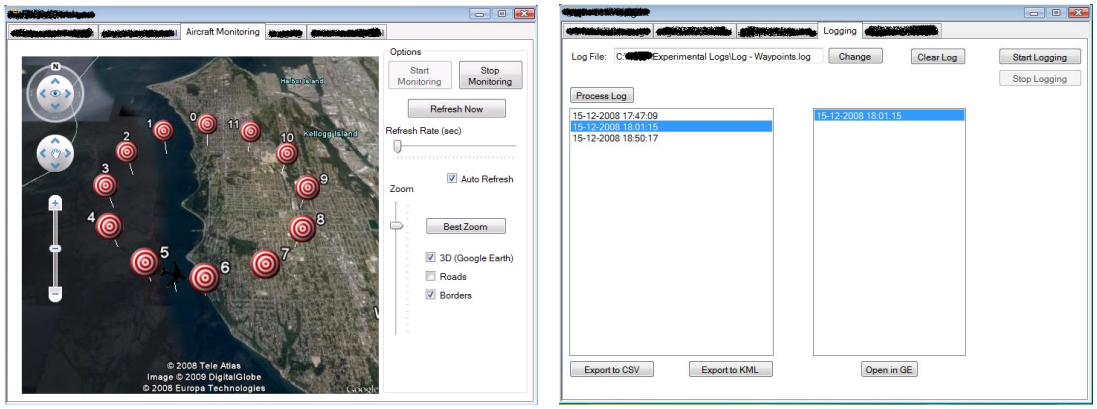
6.3.1.6 Results

The results attained through the conducted experiments, as described in the previous section, are presented here divided into two categories for a more structured arrangement – communication necessities and maneuver effectiveness.

Communication Necessities The first dimension that was analyzed is communication requirements. Although this is not a major issue when working in a simulated environment, the future use of real vehicles connected to the platform requires this to be analyzed. In this subject, the third approach is far less demanding. While both the first and second approaches need to communicate with the aircraft at regular intervals, the third approach only communicates in the beginning of the experiment and, in the case of the third scenario, two other times, due to the existence of a user-controlled loop.

In a more detailed view, the first approach needs to send elevator, throttle and aileron data at regular intervals. In the experiments that were conducted, the values were sent every second. Although this was more than enough for the simulated experiments, it is believed that this kind of approach would require, in real live, more frequent adjustments to the aircraft controls to produce a stable flight. The second approach also sent data at regular intervals, even though a higher two-second interval was used. The data sent in this approach consists of heading, speed and altitude values for the autopilot knobs. With the third approach, and considering the mentioned exception of user-controlled loops, all data is sent before-hand. This data consists of a list of a waypoint-describing structure, containing the latitude, longitude and altitude of the point, desired speed or throttle and some flags, indicating how the waypoint should be interpreted.

Maneuver Effectiveness Maneuver effectiveness was evaluated on a more subjective level, by analyzing and comparing the paths the aircrafts flew through. On a more immediate level, the experiment flights were accompanied in real-time, using both the simulator's graphical interface and the developed monitoring tool. This tool uses collected aircraft data as well as information about the determined waypoints to display the current position and orientation of the aircraft, as well as the positions and order of the various waypoints. This was done using the Google Maps and Google Earth APIs, to render the desired icons on top of the two- or three-dimensional representation of the surrounding environment in a plugin using an embedded web browser. Figure 6.14(a) shows the developed monitoring application, with the several waypoints that define a circle, and a visible aircraft between points five and six. In addition to this immediate and inaccurate visual



(a) Monitoring Tool (b) Logging Tool

Figure 6.14: Monitoring and Logging Tools

inspection of aircraft data, information about aircraft position, speed and attitude was collected at regular intervals and stored in a log file. The developed logging application then converted this information into two formats: KML format (Keyhole Markup Language), and CSV format (Comma-Separated Values), as can be seen from Fig. 6.14(b).

The first format, KML, allows for a visual inspection of the paths in an application such as Google Earth. Figures 6.15(a) and 6.15(b) show the results of two experiments conducted for the first scenario, using two distinct aircraft.

It is clear by analyzing both Fig. 6.15(a) and 6.15(b) how a faster aircraft required a change in the location of the circle and helix centers, in order to accommodate the increase of the minimum turn radius, maintaining at the same time a similar overall proportion. The CSV format can be

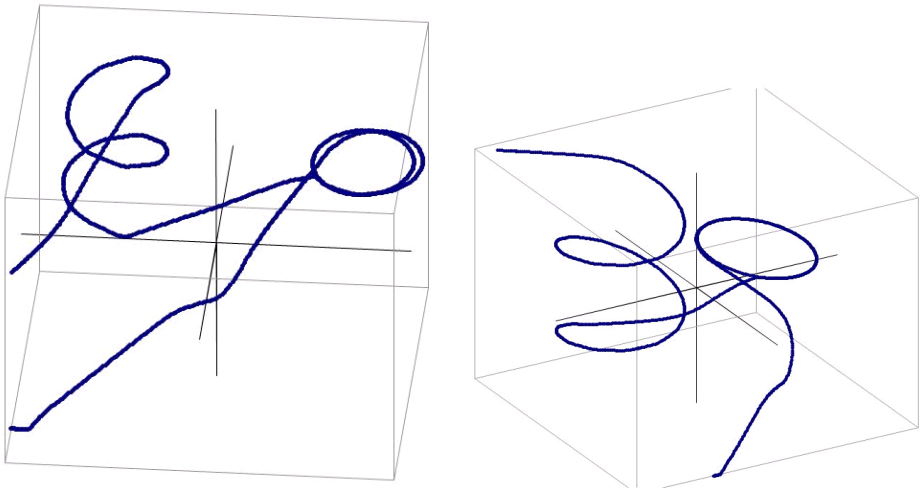
(a) Flight Path A (b) Flight Path B

Figure 6.15: Google Earth Preview of Flight Paths A and B

imported to an external application, such as Microsoft Excel, which was then used to generate three-dimensional graphs, as to facilitate in a further, more thorough, inspection of the flight paths without additional visual distractions.

Figures 6.16(a) and 6.16(b) show the results from the two experiments above in three dimensional graphs. A more detailed assessment of the efficiency of the control approaches can be

made with these graphs, especially in the altitude dimension. As previously stated, experiments



(a) 3D Graph of Flight Path A (b) 3D Graph of Flight Path B

Figure 6.16: 3D Graphs of Flight Paths A and B

were conducted with three distinct aircraft, using three distinctive scenarios and under two different weather conditions. As to assess the influence of each of these three factors in the tested approaches, the results were evaluated by grouping experiments with two common variables and analyzing the results in respect to the third one.

Considering different aircraft, and as previously stated, the second approach could not be tested in the smallest aircraft, since there is no autopilot system available to the pilot. The first aircraft had a similar performance with both the first and third approaches. The second aircraft showed a slightly better performance when using the first and third approaches, compared to the second one. The third and largest aircraft had a comparable performance for all three approaches.

Regarding the three distinct scenarios, there were no significant differences among the three approaches. The three scenarios diverged in the complexity of the intended course, each scenario adding more maneuvers and decreasing the space between maneuvering areas. This caused the aircrafts to make more abrupt turns, but all approaches handled this without posing any problem.

In respect to the influence of weather conditions in maneuver effectiveness, they were similar with all three approaches. The resulting paths were a bit more unstable, presenting more variations in altitude and in some cases widening the radius of the circling maneuvers. In the cases when this happened, the second and third approaches were more susceptible to erroneous maneuvering. In three cases, when using the third scenario, two with the second approach and one with the third approach, the aircraft performed a full circle in order to pass through a point it had previously missed. As for the first approach, the influence of deteriorating weather conditions was also felt in the form of course shifting. This was particularly visible with smaller aircraft, performing circle maneuvers, each lap slightly warped in the direction of the wind.

6.3.1.7 Conclusions

From a more theoretical point of view, one can draw some conclusions from the three approaches that were devised. The PID controller approach is a general method, which works with any aircraft that has the necessary controls (throttle, aileron and elevator). However, it requires an almost constant communication with the aircraft, in order to send the current values. Moreover, most aircraft now have some sort of autopilot system, which performs the same calculations as the PID controller, and, most probably, in a more effective manner, having the gains already tuned for the specific aircraft. The autopilot approach is not a general method, as can be seen from the chosen aircraft for the experimental activities – not all aircraft have autopilot systems. Moreover, it also requires an almost constant communication with the aircraft, just as the first approach. It does, however, require a lot less calculations to be made in the control agent side, leaving them to the autopilot system. The third approach is also a general approach, given that any aircraft can be used with the AI autopilot system. It is far less demanding in what concerns to communications requirements, since it only needs to communicate once. This is, however, a burst in communication, with far more data to be transmitted than any of the other methods. In addition to that, some substantial waypoint calculation has to be previously performed by the control agent.

From a more practical point of view, one has to consider the results that were attained through experimentation, as presented in section 6.3.1.6. In respect to maneuver execution effectiveness, there are no significant differences among the three approaches. However, and although the second approach is viable when dealing with real aircraft, it is not practical to use that approach in the simulated environment, since it would exclude some aircraft from being used. The use of different aircraft does not seem to influence the performance of the approaches, with the only visible impact being the increase of the turn radius caused by the increase of the aircraft speed. Also, all approaches seem to be equivalently affected by deteriorating weather conditions, although the autopilot-based approaches are more susceptible to missing one waypoint. This can, however, be easily corrected if one increases the distance at which the aircraft is considered to have reached a waypoint when the weather conditions deteriorate. Regarding communication requirements, one has to conclude that the third approach, based on the AI autopilot, is the best approach, since it does not require a constant communication with the aircraft, and therefore the possibility of error due to a failure in the communication with the aircraft is reduced to a minimum. The first two approaches, which had to communicate every second or once every two seconds, are more susceptible to errors caused by a communications malfunction, which could lead to an erratic maneuver performance, or even to more serious and unpredictable consequences, if the communication breakdown extents for a longer period of time. With the third approach, and even in the case of a complete communication crash, the aircraft would simply continue with the original flight plan and return to the base as originally intended to.

Considering the results obtained from the conducted experiments, the third strategy – AI autopilot – was chosen as the control strategy for vehicles within the simulator.

6.3.2 Vehicle Maneuvering Control

Maneuvering of a vehicle is performed by putting together basic maneuvers, available in an internal API. The basic maneuvers presented in section 6.3.1.4 were used, in conjunction with other higher level maneuvers and actions:

- **Go to Point**. This maneuver directs the vehicle to a specific point in space, identified by its coordinates – latitude, longitude and altitude (which, in the case of cars and boats, is always considered to be 0 ft. agl, which means it coincides with the altitude of the ground at the location of the maneuver). This basic maneuver is then used as a basis for the construction of other more complex maneuvers (given that the chosen vehicle control methodology accepts only a list of points – see section 6.3.1).

- **Circle**. In cars and boats, the altitude is always considered to be at ground level. In aircraft and submarines, the altitude can be configured, along with the center coordinates for the circle and its radius. Also, the direction of the circling motion can be specified (clockwise or counter-clockwise), as well as the number of laps (which can be fractional for just part of a circle, or unspecified for an unlimited number of laps).

- **Helix**. This maneuver is not available for cars and boats, but only for aircraft and submarines. It is similar to the circle maneuver, except that it contains both the initial and final altitudes for the helical motion. This maneuver allows aircraft and submarines to change altitude with a minimum change in the remaining coordinates.

- **Many Points**. This maneuver is comprised of a sequence of Go to Point maneuvers. It can be used when the desired maneuver does not match one of the provided shapes (circular and helical). It can be used either by land, air and water vehicles, and air vehicles can also use it for moving while on ground, during taxi operations.

- **Holding Pattern**. This maneuver is used when an aircraft requests an airport controller for landing, but the controller places it in a holding pattern, because the airport runways are being used, as already explained in section 6.2.2.1. The parameters for this maneuver are the location of the fix and the heading of the inbound leg.

- **Takeoff**. This maneuver is used by aircraft when departing from an airport. The maneuver is similar to a Many Points maneuver, except that all calculations are performed automatically, given the runway configuration and vehicle characteristics, so that the aircraft can takeoff in a smooth fashion.

- **Land**. This maneuver is similar to the previous one, but used by aircraft when landing on the airport. Point and speed calculations are also performed automatically, according to runway configuration and vehicle characteristics, as to provide a smooth and safe landing.

- **Intercept**. This maneuver can be seen as a set of maneuvers to be executed by the vehicle in order to intercept a second vehicle. The position and speed of both vehicles are taken into

account when determining the new direction and speed of the vehicle, in order to intercept the second one.

All maneuvers have the possibility of being specified along with vehicle velocity. Other maneuvering features can also be considered, such as Altitude, Heading or Speed set. Setting Altitude or Speed will effectively change current maneuvers being executed by the vehicle, so that the altitude or speed will match the desired values. Setting the desired Heading replaces currently loaded maneuvers with a sequence of maneuvers that will stabilize the vehicle traveling with the desired heading.

Maneuver control has two operational modes: online and offline. When using the online mode, maneuvers can be added directly to the ones the vehicle is currently executing, or a new maneuver can replace current maneuvers. Adding a maneuver to the ones currently being executed by the vehicle implies a small temporal overhead, due to the need to synchronize the maneuvers currently being performed by the simulator with the one being added, while replacing the current maneuvers is immediate. When using the offline mode, maneuvers are not sent to the vehicle until the synchronization process is invoked. This allows for the construction of more complex maneuvers to be made while the vehicle is still executing a previous set of maneuvers, and sent to replace them at a precise moment. This offline mode presents the advantage of not having to wait for the simulator to send data for each maneuver added to the maneuver set, thus reducing communication overheads. This offline mode is also very useful when the vehicle is being controlled by the operator, since issuing operator-controlled loops and remaining maneuvers becomes easier.

6.3.2.1 Submarine Navigation

A special attention was devoted to submarine navigation, since the simulator does not support underwater or depth simulation. As such, the Submarine Navigation Module, seen on Fig. 6.10, was developed to introduce this component. A first approach consisted in assuming a linear depth change along the submarine path, as shown in Fig. 6.17. Three waypoints (wp1, wp2 and wp3) were specified, with depths of 10, 20 and 30 meters, respectively. The depth graph on the top right corner of the image shows the depth of the vehicle, considering it is initially at the surface. The ideal or desired vehicle route would describe a curve, passing through the three waypoints, and the depth would vary in a linear fashion – in this case, three stages can be identified in the graph, corresponding to the three legs of the course before reaching each of the three waypoints.

However, this approach does not account for the fact that the simulator considers that a vehicle has reached a waypoint when the distance to the waypoint is less than a given value. This 'waypoint reached' radius has a profound impact in submarine navigation, when the previous approach is used. Figure 6.18 shows the same waypoints as considered above, and a better approximation of the actual route of the vehicle, considering the mentioned waypoint radius. As can be seen in the depth graph, on the top right corner, the vehicle has abrupt depth changes, when a waypoint is considered as reached. These fast variations are not in any way realistic, and a new approach needed to be taken.

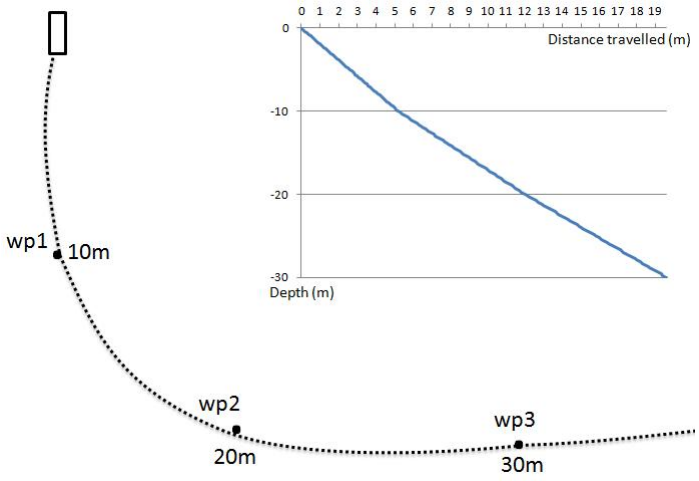


Figure 6.17: Ideal Submarine Navigation

The selected approach consists in dividing the navigation space in two regions, separated by a line that passes through the waypoint being considered in the navigation. Figure 6.19 illustrates this concept, by showing the same three waypoints as considered above, and the distance calculation steps shown for the first waypoint. First, a line is considered between the initial vehicle position and the waypoint following the current waypoint – in this case, since the vehicle is moving towards wp1, line A will connect the vehicle to wp2; then, a perpendicular line B that intersects the current waypoint is calculated; finally, a line C is computed, from the initial vehicle position toward the current vehicle position until it intersects line B; this line C is used to estimate the current depth of the vehicle by using a simple linear approach. The transition to the following iteration occurs only when the vehicle intercepts line B. In the new iteration, the initial vehicle position to be considered is the point where the vehicle intersected line B.

This method guarantees better results than the first approach on most situations. However, some exceptions exist, as is the case of a vertical motion (considering submarines that can move vertically without lateral or frontal motion). In this particular case, factors such as the submarine's maximum vertical speed need to be taken into account when computing the vehicle position. Another situation is when the vehicle is moving toward the final waypoint – in this case, since no additional waypoint exists to draw line A, line B is considered to be perpendicular to the line that connects the vehicle position when intersecting the previous waypoint to the last waypoint.

6.3.3 Agent Interaction

All interactions with other agents are performed via the agent communications platform. The interaction module is divided into smaller modules, each of which responsible for handling messages to and from a given agent type – see Fig. 6.10. This modularity helps increase code readability and maintainability. Interactions with ATC Agents were explained in detail in the previous section and interactions with the Disturbances Manager were also explained in the previous chapter.

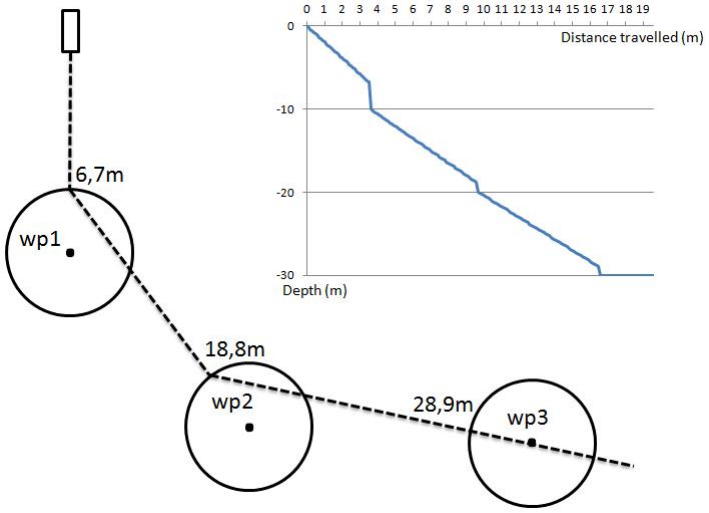


Figure 6.18: Actual Initial Submarine Navigation

Most interactions with other Vehicle Control Agents are used for planning and during mission execution.

Since the languages used to describe missions use high-level concepts, these have to be translated into lower-level constructs and concepts that can then be used to plan a given mission. These equivalences are kept in the Vehicle Control Agent, and new ones need to be coded in when new high-level concepts are introduced. Even though this may be seen as a limitation of the framework, the high-level concepts that have been described in the previous chapters for disturbances and missions are believed to represent a considerable portion of possible low-level concepts, and hence some new high-level concepts can be obtained by using lower-level constructs already present in the platform.

The initial capabilities of each vehicle, in terms of sensors and cargo, are known by every other vehicle within the team; however, the operational status of the vehicle is not known *a priori* by all teammates (a sensor may be malfunctioning, or the cargo may already have been dropped or unloaded). As such, when the mission file is loaded into the system, and transmitted to all Vehicle Control Agents (see section 3.2.2), they must determine which vehicles will participate in the mission. A simple strategy that has been implemented consists in each vehicle calculating an integer value, corresponding to the degree to which it matches the required sensors and/or cargo for the mission and its current availability to participate in the mission (which can be affected by its state, the amount of fuel it has, or by possible equipment malfunctions); a random decimal part is then added to this number, as to avoid (or at least decrease the probability of) repetitive numbers; each vehicle then broadcasts its number to the other vehicles in the team; finally, the vehicle with the highest number (the one that more closely matches the requirements for the mission) will act as the responsible vehicle for the specific mission. This vehicle will then initiate a process of selecting which vehicles will participate in the mission, according to the specifications of the mission file (see section 4.5) and the vehicles' capabilities and availabilities. A simple selection

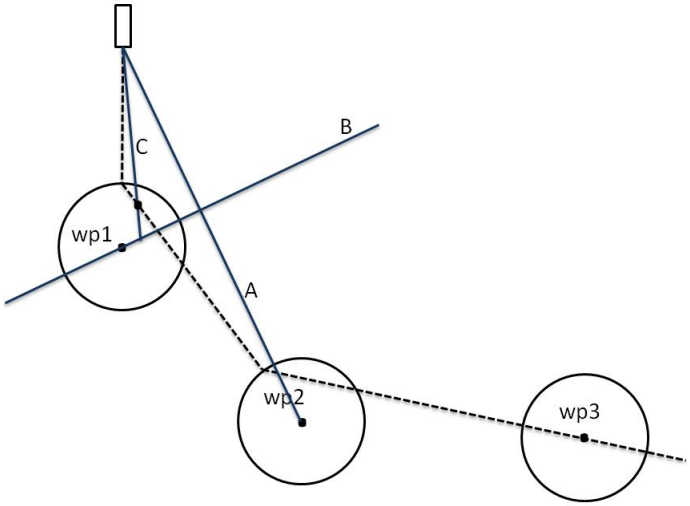


Figure 6.19: Waypoint Approach Submarine Navigation

process consists in using the Contract-Net Protocol [FIPA, 2002]. The responsible vehicle sends a call for proposals to its teammates, to which each vehicles responds with its affinity for the selected mission; finally, the responsible vehicle selects the vehicle(s) that will participate in the mission, taking into consideration mission requirements and tips regarding the assets to be used for each phase of the mission (as seen in sections 4.5 and 5.2.5). An operational difference occurs when a mission is already underway, and a second mission is sent to the team. If there are any idle vehicles, the initial step considers only those vehicles (in an attempt not to disturb the current mission); however, if the idle vehicles are not capable of performing the mission on their own, the negotiation process is extended to consider vehicles already conducting the first mission. In this case, a situation may occur when all the vehicles in the team do not have the necessary resources to execute both missions (as specified by the mission requirements, as can be seen in section 4.5.1). At this point, the operator that issued the second mission is asked to define mission priorities, and the vehicles will first execute the mission with the highest priority.

Since a mission is composed of several phases, which may be dependent on one another, when the vehicles are selected for a given mission, the first mission phase or phases to be executed need to be determined. In order to account for possible mistakes introduced by the operator in the phase type attribute (base, conditional or extra), a simple verification is performed. For that, phase predecessors (see section 4.5.1) are analyzed, and a dependency graph is built. An analysis of this dependency graph determines which phases are not dependent on any other phases. The phases selected for immediate execution are determined by cross referencing the phases with no dependencies with the ones classified as base phases.

After determining the phases to be executed first, the vehicle responsible for the mission selects the vehicles that will take part on such phases, and sends them a message to start the mission. These vehicles may be all vehicles participating on the mission, or just part of them. For instance, in a mission with two phases – detect a fire and drop water on it –, and the second one dependent on the first, different vehicles may be required for each phase – smaller vehicles equipped with

the appropriate sensors to detect the fire and larger vehicles containing the desired cargo for the second phase; in this case, only the smaller vehicles should begin the first phase, while the larger ones should wait for a fire to be detected before departing from the base of operations. The selection of the vehicles that should participate in the initial phases also takes into consideration the requirements and tips for these phases.

The current implementation of the Vehicle Control Agent does not contemplate all items included in the phase tips. In more detail, formations, search pattern, search area division method and strategy are not yet implemented. There are several approaches to formation movement using several vehicles, both in two- and three-dimensional environments (see section 2.2.3). The coding modularity used in the platform allows for the possibility to code several approaches to formation flying and use one of them selectively. This allows the platform to be used for this research niche, comparing several methods and possibly helping in the development and analysis of a new methods. Several search patterns can be found in literature regarding search & rescue operations (see sections 1.3 and 4.5.2). The implementation of a search pattern for a single vehicle for a given mission area can be achieved by using several waypoint calculation methods (each associated with a given search pattern) that use the available vehicle basic maneuvers (see section 6.3.2). However, it is desirable to combine a given search pattern with a given formation, so that several vehicles can execute the search in an effective manner. Also, the search area division method to be used can be chosen from one of several methods that have been presented by the community [Schneider-Fontán & Matarić, 1998] [Jäger & Nebel, 2002]. The strategy to be used by the team (tactics and activating conditions) may be implemented in the near future, using activation conditions determined by a set of methods that subscribe to information from the simulator and raise events when such conditions are met. Another method subscribing to these events will determine the appropriate actions to be taken, according to the currently active conditions and the strategy definition.

6.3.4 Summary

In this section, the Vehicle Control Agent was described in more detail, and its internal architecture was introduced, as well as the interface used for development and tests. First, a major implementation decision regarding the manner in which the vehicles are controlled within the simulator was presented. The three considered approaches were described in detail, and the experiments that were conducted in order to choose the approach that optimizes communication requirements and maneuver effectiveness were presented, along with an analysis of the obtained results. Then, the set of available high-level maneuvers used to control the vehicles was presented, and some implementation details regarding submarine navigation (which the simulator provides no support for) were described. Finally, some information regarding interaction among Vehicle Control Agents is presented, focusing on how vehicles are assigned to missions and phases.

The following section describes the agent communication platform used to facilitate communication among agents.

6.4 Conclusions

This chapter presented an overview on the Agent Communication Platform and also the two autonomous entities of the developed platform – the ATC Agent and the Vehicle Control Agent.

First, section 6.1 presented a few considerations regarding the chosen agent communication platform, and described how some of its services are used.

Before introducing the ATC Agent, an overview on ATC operations was presented, along with a brief description of some work that has been developed for both real ATC operations and traffic control. Then, the ATC Agent was described, detailing several implementation details, namely the interactions with the Vehicle Control Agent, and the specifics of ATC protocols and operations. The results on the conducted experiments were also presented, which both validated the approach and exposed several possibilities for increasing the performance of this agent.

Section 6.3 described the Vehicle Control Agent, providing with an overview of its internal organization, and detailing several modules – first, a detailed analysis was presented, describing how this agent and the chosen simulator interact with each other, so that the Vehicle Control Agent can control any vehicle within FSX; then, the maneuvering control module is outlined, describing the existing high-level maneuvers and how these are mapped into the simulator, and also detailing the specifics of submarine navigation; finally, a brief description of interactions among Vehicle Control Agents is presented.

This chapter, allied with the previous one, and the overview of the platform architecture, presented in section 3.2, provides with a better idea of the potential of this platform in several research areas.

Chapter 7

Conclusions and Future Work

This chapter provides with an overview of the achieved results, containing a summary of the contributions presented herein, their originality and limitations, as well as several lines for improvements and future developments.

A platform was designed and developed that can support the execution of several missions by a group of autonomous vehicles. The design of the platform's architecture, as presented in chapter 3, brought forward a secondary contribution, concerning the design of systems that include mobile robotic vehicles – the generic model presented in section 3.1 can significantly reduce most of the common design tasks and implementation difficulties, while at the same time providing a high-level overview of the system as a whole.

The first task after designing the platform architecture, the choice of a simulator, also led to the construction of an evaluation method for simulators, as presented in section 5.1.1. As such, four main categories were used to evaluate the simulators (simulation engine, graphics, fault injection and openness), each of which evaluating several features of the simulator. In addition to the four considered categories, a fifth category can be considered, which may include the cost of the simulator (often considered when comparing different platforms) and any additional idiosyncratic simulator features. This process led to the choice of Flight Simulator X as the simulator for the platform.

Besides the simulator, the platform includes several other components, as presented in chapters 5 and 6. Some of these components provide the foundations to carry out research in several specific areas, such as centralized traffic control or performance analysis in multi-agent systems with autonomous vehicles.

The Control Panel constitutes the main interface between the platform and the operator, allowing for the configuration of most of the aspects regarding both the platform and mission specification. Several usability concerns were taken into consideration when developing this interface component, and several improvements can be thought of, which would most likely increase usability and decrease the time taken to configure a mission.

The Disturbances Manager simulates the behavior of elements anomalous to the environment, and communicates both with the simulator and with Vehicle Control Agents, simulating vehicle sensor readings when in sensing range of a disturbance.

The Monitoring Tool allows for both a macro and micro vision of the simulation session, presenting a graphical overview of the team vehicles, and also the possibility to monitor, in real-time, several simulation parameters for each vehicle.

The Logging Mechanism generates several files for each simulation, which allows for both a mission replay and analysis, by the Performance Analysis Tool. This tool, based on these log files, evaluates the performance of the team when executing the mission. This performance evaluation is determined according to a profile, defined by the operator, which is constituted by several metrics, which can relate to either a single vehicle or all team vehicles. It also allows for several sessions to be combined, to produce average results in order to mitigate the effect of possible outliers, and to compare simulation sessions (thus comparing different algorithms, methods or strategies used in the different sessions).

The ATC Agent directs traffic on and around a base of operations, in a centralized manner, allowing for research to be conducted over centralized vs. decentralized traffic control, and also to compare several traffic control algorithms.

The Vehicle Control Agent constitutes one of the main components of the platform, each instance representing a vehicle in the team. It is responsible for mission planning and several other aspects regarding mission execution. Vehicle maneuvering within the simulator was implemented taking into account the specifics of the simulator, including the need to develop a module for submarine navigation.

Four languages (Scenario, Team, Disturbances and Mission) were developed in order to allow for a detailed and flexible configuration of the several elements that altogether enable the mission to be performed with the specified criteria. Classified as Static vs. Dynamic and Environment- vs. Team-Oriented, these languages allow for the specification of almost every aspect involved in the simulation, from the physical scenario in which the mission will take place and the disturbances that exist in that scenario to the definition of team composition and capabilities, and the definition of the mission to be executed. SDL describes a set of bases of operations, global control structures (traffic controllers and no-fly areas) and defines existing vehicle types. DDL defines the vehicles that compose a team, as well as team-specific restrictions, such as the bases of operations it can use, or additional no-fly areas. DDL specifies all disturbances in the environment, and their behavior details. Finally, MDL allows for the specification of a mission comprised of several phases, each with possible requirements and soft constraints, as well as targets (on which actions of different classes can be performed). These languages allow for a flexible specification of several elements:

- By combining SDL and TDL, vehicles can be created and simulated that are characterized by heterogeneity at several levels – vehicle type (aircraft, ground, water or underwater vehicles), characteristics (cruise and maximum speeds, fuel consumption, aircraft operational altitude and range, among several others) and sensors and/or cargo they transport.

- Using DDL, several types of disturbances can be specified – living beings (such as people or animals), natural occurring phenomena (such as fires, or hydrothermal vents) and human-induced events (such as pollution), or vehicles (be it water, air or land vehicles). The flexibility of DDL allows for disturbances to be created in either a specific or random point of space and time; disturbances can be either stationary or mobile (including a number of distinct motion patterns), with a variable number of components, each with fixed or variable size (including dispersing patterns), a possible simulation representation, and requiring different sensors to be detected.

- Using MDL, several mission classes can be identified. As stated in section 5.6, these include Search (Detect), Detect Origin, Measure, Follow, Load/Unload and Transport. Some of these mission classes (namely the Detect, Detect Origin and Follow) can be further detailed, according to the classification of the target:

  - **Multiplicity (Single vs. Multiple)**. There can be only one target to be detected/followed, or there can be multiple targets.
  - **Motion (Stationary vs. Mobile)**. The target can either be motionless, remaining at a fixed location, or it can be mobile, its location changing throughout the simulation wither on a predicted path or in a random manner.
  - **Size (Constant vs. Variable)**. The size of a target can remain the same throughout its existence, or it can change over time.
  - **Detection (Simple vs. Combined)**. The target may require only one sensor to be detected (in which case one vehicle is enough to detect the target), or it may require several sensor reading to be combined in order to confirm the presence of the target (in which case one vehicle may not be enough for detection).

  This classification can be used to determine a relative difficulty level of the mission being performed by the team. This information can also be useful when creating metric profiles to evaluate a given mission – a profile designed to evaluate a mission with only one stationary target, with constant size and simple detection will probably differ from a profile designed to evaluate a mission with multiple mobile targets with variable size and requiring a combined detection.

These languages provide with a means to configure scenario and disturbances to it, as well as team composition and missions, in a systematic manner. This circumvents the use of ad-hoc configurations or specific coding, making them not only a powerful configuration tool for multi-vehicle systems, but also a good candidate to standardize scenario, team, disturbances and mission definition in similar platforms.

By using XML, several desired characteristics were achieved – extensibility, data validation and system-independence. In fact, each of the four languages can easily be extended by editing the corresponding Schema, and with the aid of automated tools, namely, the XML Schema Definition

Tool, the developed applications can rapidly be altered to meet the changes made to the languages. Data validation is also achieved by the use of XML Schema, so that both structure and content can be validated. Also, the languages are easily read by both humans and machine, and its content understood by any human, expert on the domain or not. Most of the concepts present in the developed languages are high-level concepts, which promotes their use by non-experts.

7.1 Summary of Contributions and Limitations

As described above, several original contributions can be found throughout this document:

- The general model for multi-robot systems presented in section 3.1.
- The specific implementation of the general architecture, as presented in section 3.2, with the use of a simulator that allows for a complex environment to be simulated and provides a realistic visual feedback [Silva et al., 2009b] and several other components.
- The high-level languages for the definition of scenario, teams, disturbances and missions described in chapter 4, along with their classification as scenario- vs. team-oriented and static vs. dynamic.

A user manual has also been written, as to allow a quick familiarization with the developed platform [Silva, 2011]. It contains an overall description of the platform, an installation guide for the several components (Flight Simulator X, AgentService and registry configuration and other necessary components), a detailed guide for using the several components, with a focus on the Control Panel, and also a development guide, describing the implementation details of the several components and guidelines for future developments.

The limitations of this work are visible in the current implementation since some functionalities are only available for aircraft, but not fully functional for ground or water and underwater vehicles – for instance, the ATC Agent can currently only handle airports and air traffic.

7.2 Future Work

Some extensions and new functionalities can be identified to constitute future work lines, based on the developed platform:

- Expansion of the ATC Agent for land and water traffic, in ports and ground bases. This expansion is deemed to be rather simple, given that the same data structures are used to describe roads, waterways and the airport taxiway network.
- One of the expansions that could be implement is related to failure. One module would be developed to be responsible for injecting failures into existing vehicles (either immediately or at pre-programmed times) A second module would be responsible for monitoring the vehicle status, and detecting failures with the vehicle. The vehicle would then have to make operational decisions based on the type of failure.

- Expansion to the real world, by developing external modules in order to communicate with actual vehicles. This would allow for the framework to function in a virtual, augmented or real environment. The architecture already foresees this possibility and asides the external module, it only requires some adaptations to the Vehicle Control Agent.
- Extend the implementation of the Disturbances Manager to consider all possibilities currently allowed by DDL. Also, more dispersion models and growth predefinitions can be added to the DDL definition and to the disturbances section of the Control Panel.
- Another feature that can be implemented pertains to simulation using multiple simulators – this subject was already mentioned in section 5.1.2.1. In order to use multiple simulators, almost every module of the platform must be adapted, in order to support multiple connections to the various simulator instances. Furthermore, the Vehicle Control Agent will have to be able to perform the change from one simulator to the other, when crossing simulator coverage areas.
- In respect to the agent communication platform, a task that could be done consists in changing the model for message reception. While the current model is based on polling, it may be desired that message reception be based on an event being triggered when the message is sent to the destination. Other tasks that can be thought of, when considering this platform, are improving the working of the platform in federation mode, and the security and efficiency of communications.
- The possibility to import airport files directly from FSX scenario files (such as tools like ADE and AFX do), and allow for a graphical interface for the construction of airports, ports and ground bases to be used in the Control Panel.
- A complete description of the road system, as well as maritime and aerial routes (or at least the corridors for the airports) could also be included, either as information explicitly present on the files, or as a specification of a source for the information, such as a GIS-based database or an external service [Huang et al., 2009].
- Related to the previous feature, and when considering vehicles or persons, an additional improvement to DDL would be to consider some degree of intelligence in the vehicle/person course planning. For instance, instead of providing the entire path for a vehicle to traverse, the user could simply specify the departure and arrival points; based on the road network topology (and possibly taking into consideration current traffic conditions, which would require a connection to a real-time traffic information system), the vehicle would then determine the best course to reach its destination.
- The use of disturbance templates is also a possibility, as to allow for the automatic filling of several details of a disturbance definition, given its type. For instance, when creating a new fire, part of the fields could be filled, and standard fire components automatically generated with standard information.

- Additional features could also be considered, such as the activation of a disturbance by a trigger. For instance, and considering a military context, one popular mission is known as Wide-Area Search & Destroy (WASD), where a team of Unmanned Aerial Vehicles (UAV) has to fly over a given area in search for an enemy target (as is the case of surface-to-air missile (SAM) bases), and destroy it after a positive identification has been made [Stolarik, 2007]. In this context, it is often the case that SAMs are fired when a vehicle is detected, thus increasing the difficulty of the mission. This could be attained by introducing a new start type in the disturbance availability element – a trigger (which, in this case, would be the presence of a vehicle within the detection range of the SAM). This would also require that disturbances and vehicles interact at another level, and vehicles to be able to suffer damages (this requires the failure-related feature described above to be implemented).

In sum, this dissertation introduced an architecture for a platform that allows for multi-robot missions to be performed autonomously. These missions are configured using four description languages – scenario, teams, disturbances and mission – which are categorized as static vs. dynamic and scenario- vs. team-oriented. The developed platform not only implements the proposed architecture but is also flexible and modular enough as to allow for several developments to be added to it and can also act as a research platform for several specific areas.

References

[Abreu et al., 2007] Abreu, P., Mendes, P., Silva, D. C., & Vinhas, V. (2007). MicroCredit Practices Applied to e-Supply Chain Management. In S. Krishnamurthy & P. Isaías (Eds.), *Proceedings of the IADIS International Conference on e-Commerce 2007, December 7–9 2007, Algarve, Portugal* (pp. 91–98).

[Alexander et al., 2005] Alexander, A. L., Brunyé, T., Sidman, J., & Weil, S. A. (2005). From Gaming to Training: A Review of Studies on Fidelity, Immersion, Presence, and Buy-in and their Effects on Transfer in PC-Based Simulations and Games.

[Ali et al., 2010] Ali, G., Shaikh, N. A., Shah, M. A., & Shaikh, Z. A. (2010). Decentralized and Fault-Tolerant FIPA-Compliant Agent Framework Based on .Net. *Australian Journal of Basic and Applied Sciences*, 4(5), 844–850.

[Allen et al., 2009] Allen, S., Burke, E. K., Hyde, M. R., & Kendall, G. (2009). Evolving Reusable 3D Packing Heuristics with Genetic Programming. In F. Rothlauf (Ed.), *Proceedings of the 11th Annual Genetic and Evolutionary Computation Conference(GECCO 2009), July 8–12 2009, Montreal, Québec, Canada* (pp. 931–938).: ACM.

[Alm, 2002] Alm, A. (2002). *Autopilot for Autonomous Ground Vehicle, Modelling, Design and Implementation*. Technical Report FOI-R–0770–SE, Swedish Defense Research Agency – System Technology Division, Stockholm, Sweden.

[Aloisio et al., 2006] Aloisio, G., Conte, D., Elefante, C., Epicoco, I., Marra, G. P., Mastrantonio, G., & Quarta, G. (2006). SensorML for Grid Sensor Networks. In H. R. Arabnia (Ed.), *Proceedings of the 2006 International Conference on Grid Computing & Applications, GCA 2006, Las Vegas, Nevada, USA, June 26-29, 2006* (pp. 147–152).: CSREA Press.

[Altarriba et al., 2003] Altarriba, J., Basnight, D. M., & Canary, T. M. (2003). Emotion Representation and Perception Across Cultures. In W. J. Lonner, D. L. Dinnel, S. A. Hayes, & D. N. Sattler (Eds.), *Online Readings in Psychology and Culture (Unit4 – Basic Psychological Processes and Culture)* chapter 5. Center for Cross-Cultural Research, Western Washington University.

[Alur et al., 2000] Alur, R., Grosu, R., Hur, Y., Kumar, V., & Lee, I. (2000). Modular Specification of Hybrid Systems in CHARON. In N. A. Lynch & B. H. Krogh (Eds.), *Proceedings of the 3rd International Workshop on Hybrid Systems: Computation and Control (HSCC 2000), March 23–25 2000, Pittsburgh, Pennsylvania, USA*, volume 1790 of *Lecture Notes in Computer Science* (pp. 6–19).: Springer.

[ANSI/AIIM, 2009] ANSI/AIIM (2009). *Standard Recommended Practice - Strategy Markup Language – Part 1: StratML Core*. Standard ANSI / AIIM 21, American National Standards Institute / Association for Information and Image Management.

[AOS, 2005a] AOS (2005a). *JACK Agent Manual*. Agent Oriented Software (AOS) Pty. Ltd., Victoria, Australia, 5.3 edition.

[AOS, 2005b] AOS (2005b). *JACK Teams Manual*. Agent Oriented Software (AOS) Pty. Ltd., Victoria, Australia, 5.3 edition.

[Arunachalam & Sadeh, 2005] Arunachalam, R. & Sadeh, N. M. (2005). The Supply Chain Trading Agent Competition. *Electronic Commerce Research Applications*, 4(1), 66–84.

[Balakirsky et al., 2006] Balakirsky, S., Scrapper, C., Carpin, S., & Lewis, M. (2006). USARSim: Providing a Framework for Multi-Robot Performance Evaluation. In *Proceedings of the 6th Workshop on Performance Metrics for Intelligent Systems (PerMIS 2006), August 21–23 2006, Gaithersburg, Maryland, USA* (pp. 98–102).

[Balakirsky et al., 2007] Balakirsky, S., Scrapper, C., Carpin, S., & Lewis, M. (2007). USARSim: a RoboCup Virtual Urban Search and Rescue Competition. In G. R. Gerhart, D. W. Gage, & C. M. Shoemaker (Eds.), *Proceedings of the SPIE Unmanned Systems Technology IX, Defense and Security Symposium, April 9 2007, Orlando, Florida, USA*, volume 6561.

[Balch & Arkin, 1994] Balch, T. R. & Arkin, R. C. (1994). Communication in Reactive Multiagent Robotic Systems. *Autonomous Robots*, 1(1), 27–52.

[Ballard, 1977] Ballard, R. D. (1977). Notes on a Major Oceanographic Find. *Oceanus*, 20(3), 35–44.

[Ballard, 1993] Ballard, R. D. (1993). The MEDEA/JASON Remotely Operated Vehicle System. *Deep Sea Research Part I: Oceanographic Research Papers*, 40(8), 1673–1687.

[Baross & Hoffman, 1985] Baross, J. A. & Hoffman, S. E. (1985). Submarine Hydrothermal Vents and Associated Gradient Environments as Sites for the Origin and Evolution of Life. *Origins of Life and Evolution of Biospheres*, 15(4), 327–345.

[Barratt, 2001] Barratt, R. (2001). *Atmospheric Dispersion Modelling: An Introduction to Practical Applications*. Business and the Environment: Practitioner. Earthscan Publications Limited, 1st edition.

[Bauer et al., 2001] Bauer, B., Müller, J. P., & Odell, J. (2001). Agent UML: A Formalism for Specifying Multiagent Software Systems. *International Journal of Software Engineering and Knowledge Engineering*, 11(3), 207–230.

[Bayer & Svantesson, 2001] Bayer, P. & Svantesson, M. (2001). Comparison of Agent-Oriented Methodologies: Analysis and Design – MAS-CommonKADS versus Gaia. In *Student Workshop on Agent Programming*: Blekinge Institute of Technology.

[Bellifemine et al., 2007] Bellifemine, F., Caire, G., & Greenwood, D. (2007). *Developing Multi-Agent Systems with JADE*. Wiley Series in Agent Technology. John Wiley & Sons, Ltd.

[Bellifemine et al., 1999] Bellifemine, F., Poggi, A., & Rimassa, G. (1999). JADE - A FIPA-Compliant Agent Framework. In H. Nwana & D. Ndumu (Eds.), *Proceedings of the 4th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'99), April 19–21 1999, London, UK* (pp. 97–108).: The Practical Application Company.

[Benini, 2004] Benini, E. (2004). Significance of Blade Element Theory in Performance Prediction of Marine Propellers. *Ocean Engineering*, 31(8–9), 957–974.

[Bennett et al., 1996] Bennett, C. L., Carter, M. R., & Fields, D. J. (1996). *Detection of Illegal Drugs using Passive Infrared Sensing*. Technical Report UCRL-ID–123847, U.S. Department of Energy (DoE), Office of Scientific and Technical Information (OSTI), Tennessee,USA.

[Bergenti et al., 2004] Bergenti, F., Gleizes, M.-P., & Zambonelli, F., Eds. (2004). *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*. Multiagent Systems, Artificial Societies, and Simulated Organizations: International Book Series. Kluwer Academic Publishers, 1 edition.

[Berndt et al., 2008] Berndt, J. S., Peden, T., Culp, D., Megginson, D., Hofman, E., Frölich, M., Marco, A. D., Luff, D., Duke, L., Galbrait, B., Jackson, B., & Olson, C. (2008). *JSBSim - An Open Source, Platform-Independent, Flight Dynamics Model in C++*. JSBSim Development Team, 1.0 edition.

[Bird, 2005] Bird, S. (2005). You're Cleared for Take-off - How to Tackle a Fear of Flying on a Virtual Plane. The Times. Available online at `http://www.timesonline.co.uk/tol/life_and_style/health/features/article549165.ece` (accessed December 2009).

[Blidberg, 2001] Blidberg, D. R. (2001). The Development of Autonomous Underwater Vehicles (AUV); a Brief Summary. Autonomous Undersea Systems Institute, Lee New Hampshire, USA.

[Boccalatte et al., 2004] Boccalatte, A., Gozzi, A., Grosso, A., & Vecchiola, C. (2004). AgentService. In F. Maurer & G. Ruhe (Eds.), *Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering (SEKE'2004), June 20–24 2004, Banff, Alberta, Canada,* (pp. 45–50).

[Bond & Gasser, 1988] Bond, A. H. & Gasser, L. G. (1988). *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers, Inc.

[Braga, 2010] Braga, R. A. M. (2010). *Intelligent WheelChair Development Platform (in Portugues: Plataforma de Desenvolvimento de Cadeiras de Rodas Inteligentes)*. PhD thesis, Faculty of Engineering, University of Porto, Porto, Portugal.

[Bresciani et al., 2002] Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., & Perini, A. (2002). *Tropos: An Agent-Oriented Software Development Methodology*. Technical Report 0212-82, Istituto Trentino di Cultura.

[Bresciani et al., 2004] Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., & Mylopoulos, J. (2004). Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3), 203–236.

[Brooks, 1986] Brooks, R. A. (1986). A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2(1), 14–23.

[Brown et al., 2001] Brown, J., Montgomery, K., Latombe, J.-C., & Stephanides, M. (2001). A Microsurgery Simulation System. In W. J. Niessen & M. A. Viergever (Eds.), *Proceedings of the 4th International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI 2001), October 14–17 2001, Utrecht, The Netherlands*, volume 2208 of *Lecture Notes in Computer Science* (pp. 137–144).: Springer Berlin / Heidelberg.

[Brunk & Porosnicu, 2005] Brunk, B. & Porosnicu, E. (2005). Aeronautical Information Exchange Model (AIXM) GIS Interoperability through GML. In *Proceedings of the 25th Annual ESRI International User Conference, July 25–29 2005* (pp. 17 pages).

[Brunk & Porosnicu, 2004] Brunk, B. K. & Porosnicu, E. (2004). A Tour of the AIXM Concepts. In *24th Annual ESRI International User Conference, August 9–13 2004* (pp. 9 pages).

[Brunner et al., 2007] Brunner, H., Mikula, A., & Eier, D. (2007). A Concept for Service Based Information Quality and Safety Enhancement in Aeronautical Information Management. In *Proceedings of the 52nd Annual Conference of the Air Traffic Control Association 2007, 28-31 October 2007, Washington, Dc.* (pp. 43–47).: Air Traffic Control Association (ATCA) Curran Associates, Inc.

[Buehler et al., 2009] Buehler, M., Iagnemma, K., & Singh, S., Eds. (2009). *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, volume 56 of *Springer Tracts in Advanced Robotics*. Springer, 1st edition.

[Callantine, 2002] Callantine, T. J. (2002). *CATS-based Air Traffic Controller Agents*. Contractor Report NASA/CR-2002-211856, National Aeronautics and Space Administration, Ames Research Center, Moffett Field, California, USA.

[Callantine et al., 1999] Callantine, T. J., Mitchell, C. M., & Palmer, E. A. (1999). *GT-CATS: Tracking Operator Activities in Complex Systems*. Technical Memorandum NASA/TM-1999-208788, National Aeronautics and Space Administration, Ames Research Center, Moffett Field, California, USA.

[Cantoni & Neto, 2008] Cantoni, L. A. & Neto, A. A. (2008). Hardware-in-the-Loop with Microsoft Flight Simulator. in Portuguese (original title: Hardware-in-the-Loop com o Simulador de Vôo Microsoft Flight Simulator).

[Cao & Sun, 2008] Cao, J. & Sun, C. (2008). A Mission Planning System for an Autonomous Underwater Vehicle. In *Proceedings of the 7th World Congress on Intelligent Control and Automation (WCICA 2008), June 25–27 2008, Chongqing, China* (pp. 3915–3919).

[Casbeer et al., 2006] Casbeer, D. W., Kingston, D. B., Beard, R. W., & McLain, T. W. (2006). Cooperative Forest Fire Surveillance Using a Team of Small Unmanned Air Vehicles. *International Journal of Systems Science*, 37(6), 351–360.

[Casper & Murphy, 2003] Casper, J. & Murphy, R. (2003). Human-Robot Interactions during the Robot-Assisted Urban Search and Rescue Response at the World Trade Center. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 33(3), 367–385.

[Castro & Oliveira, 2008] Castro, A. & Oliveira, E. (2008). The Rationale Behind the Development of an Airline Operations Oontrol Centre using Gaia-based Methodology. *International Journal of Agent-Oriented Software Engineering*, 2(3), 350–377.

[Cernuzzi et al., 2004] Cernuzzi, L., Juan, T., Sterling, L., & Zambonelli, F. (2004). The Gaia Methodology - Basic Concepts and Extensions. In F. Bergenti, M.-P. Gleizes, & F. Zambonelli (Eds.), *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, volume 11 of *Multiagent Systems, Artificial Societies, and Simulated Organizations* chapter 4, (pp. 69–88). Springer US.

[Cernuzzi & Zambonelli, 2004] Cernuzzi, L. & Zambonelli, F. (2004). Experiencing AUML in the Gaia Methodology. In *Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS'04), April 14–17 2004, Porto, Portugal*, volume III - Information Systems Analysis and Specification (pp. 283–288).: Kluwer Academic Publisher.

[Chao et al., 2007] Chao, H., Cao, Y., & Chen, Y. (2007). Autopilots for Small Fixed-Wing Unmanned Air Vehicles: A Survey. In *Proceedings of the 2007 IEEE International Conference on Mechatronics and Automation (ICMA 2007), August 5–8 2007, Harbin, Heilongjiang, China* (pp. 3144–3149).: IEEE.

[Chavent et al., 2007] Chavent, M., Guegan, H., Kuentz, V., Patouille, B., & Saracco, J. (2007). Pollution sources detection via principal component analysis and rotation. In *Proceedings of the XIIth International Symposium of Applied Stochastic Models and Data Analysis (ASMDA 2007), Chania, Crete, Greece*.

[Ciancarini, 1996] Ciancarini, P. (1996). Coordination Models and Languages as Software Integrators. *ACM Computing Surveys*, 28(2), 300–302.

[Cimorelli et al., 2004] Cimorelli, A. J., Perry, S. G., Venkatram, A., Weil, J. C., Paine, R. J., Wilson, R. B., Lee, R. F., Peters, W. D., Brode, R. W., & Paumier, J. O. (2004). *AERMOD: Description of Model Formulation*. Technical Report EPA-454/R-03-004, U.S. Environmental Protection Agency, Office of Air Quality Planning and Standards, Emissions Monitoring and Analysis Division, North Carolina, USA.

[CloudCap Technology, 2008] CloudCap Technology (2008). *A Brief History of Piccolo Development*. CloudCap Technology, Hood River, Oregon, USA.

[Cohen et al., 1990] Cohen, P. R., Levesque, H. J., Nunes, J. H. T., & Oviatt, S. L. (1990). Task-Oriented Dialogue as a Consequence of Joint Activity. In T. Fukumura (Ed.), *Proceedings of the 1st Pacific Rim International Conference on Artificial Intelligence (PRICAI '90), November 14–16 1990, Nagoya, Japan* (pp. 203–208).: Ohmsha, Ltd.

[Coifman et al., 2004] Coifman, B., McCord, M., Mishalani, R. G., & Redmill, K. (2004). Surface Transportation Surveillance from Unmanned Aerial Vehicles. In *Proceedings of the 83rd Annual Meeting of the Transportation Research Board, January 11–15 2004, Washington D.C., USA* (pp. 11–20).

[Collins et al., 2006] Collins, J., Arunachalam, R., Sadeh, N., Eriksson, J., Finne, N., & Janson, S. (2006). *The Supply Chain Management Game for the 2007 Trading Agent Competition*. Technical Report CMU-ISRI-07-100, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA.

[Collins et al., 2000] Collins, J., Ndumu, D., & van Buskirk, C. (2000). *The Zeus Technical Manual*. British Telecom Intelligent Systems Research Centre, England, 1.04 edition.

[Collis & Ndumu, 2000] Collis, J. & Ndumu, D. (1999–2000). *ZEUS Methodology Documentation Parts I, III and IV: The Role Modelling Guide, the Application Realisation Guide and The Runtime Guide*. Technical report, British Telecom Intelligent Systems Research Centre, England.

[Contreras et al., 2004a] Contreras, M., Germán, E., Chi, M., & Sheremetov, L. (2004a). Design and Implementation of a FIPA Compliant Agent Platform in .NET. In V. Skala & P. Nienaltowski (Eds.), *Proceedings of the 2nd International Workshop on .NET Technologies, May 31–June 2 2004, Plzen, Czech Republic* (pp. 9–16).: UNION Agency - Science Press.

[Contreras et al., 2004b] Contreras, M., Germán, E., Chi, M., & Sheremetov, L. (2004b). Design and Implementation of a FIPA Compliant Agent Platform in .NET. *Journal of Object Technology Special issue: .NET Technologies 2004 workshop,*, 3(9), 5–28.

[Coumou et al., 2006] Coumou, D., Driesner, T., Geiger, S., Heinrich, C. A., & Matthäi, S. (2006). The Dynamics of Mid-Ocean Ridge Hydrothermal Systems: Splitting Plumes and Fluctuating Vent Temperatures. *Earth and Planetary Science Letters*, 245(1–2), 218–231.

[Craighead et al., 2007] Craighead, J., Murphy, R. R., Burke, J., & Goldiez, B. F. (2007). A Survey of Commercial & Open Source Unmanned Vehicle Simulators. In *IEEE International Conference on Robotics and Automation 2007 (ICRA 2007), April 10–14 2007, Rome, Italy* (pp. 852–857).: IEEE.

[Crichton et al., 2001] Crichton, J. W., Koslow, S., Morris, J., & Owens, J. (2001). *Alternate Route: An Approach to Enhancing ATC System Capacity*. Nav Canada, Ontario, Canada.

[Cruz & Alves, 2008a] Cruz, N. A. & Alves, J. C. (2008a). Autonomous Sailboats: An Emerging Technology for Ocean Sampling and Surveillance. In *Proceedings of the 2008 MTS/IEEE OCEANS Conference and Exhibition, September 15–18 2008, Quebec, Canada* (pp. 6 pages).

[Cruz & Alves, 2008b] Cruz, N. A. & Alves, J. C. (2008b). Ocean Sampling and Surveillance using Autonomous Sailboats. In *Proceedings of the 1st International Robotic Sailing Comference (IRSC 2008), May 23–24 2008, Breitenbrunn, Austria* (pp. 30–36).

[da Costa et al., 2008] da Costa, R. T., Sardinha, A., & Nardi, A. E. (2008). Virtual Reality Exposure in the Treatment of Fear of Flying. *Aviation, Space, and Environmental Medicine*, 79(9), 899–903.

[Dam & Winikoff, 2003] Dam, K. H. & Winikoff, M. (2003). Comparing Agent-Oriented Methodologies. In P. Giorgini, B. Henderson-Sellers, & M. Winikoff (Eds.), *Proceedings of the 5th International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS 2003), Revised Selected Papers, July 14 2003, Melbourne, Australia and October 13 2003, Chicago, Illinois, USA*, volume 3030 of *Lecture Notes in Computer Science* (pp. 78–93).: Springer.

[Danek, 1993] Danek, G. L. (1993). *Vertical Motion Simulator Familiarization Guide*. Technical Memorandum 103923, NASA Ames Research Center, Moffett Field, California, USA.

[Das et al., 1999] Das, Y., Russell, K. L., Kircanski, N., & Goldenberg, A. A. (1999). An Articulated Robotic Scanner for Mine Detection – a Novel Approach to Vehicle Mounted Systems. In A. C. Dubey, J. F. Harvey, J. T. Broach, & R. E. Dugan (Eds.), *Detection and Remediation Technologies for Mines and Minelike Targets IV* (pp. 1396–1404).: SPIE (Society of Photo-Optical Instrumentation Engineers).

[Dàvila-Rìos et al., 2008] Dàvila-Rìos, I., Torres-Trevino, L. M., & Lòpez-Juàrez, I. (2008). On the Implementation of a Robotic Welding Process Using 3D Simulation Environment. In *Proceedings of the 2008 Electronics, Robotics and Automotive Mechanics Conference (CERMA'08), September 30– October 3 2008, Cuernavaca, Morelos, Mexico* (pp. 283–287).: IEEE Computer Society.

[de Farias et al., 2007] de Farias, R. C., de Almeida, G. F., Teichrieb, V., & Kelner, J. (2007). Flight Instructor, a Virtual Flight Instructor for Microsoft Flight Simulator X. In M. W.

de Souza Ribeiro & W. A. da Silva (Eds.), *Proceedings of the IV Workshop on Virtual and Augmented Reality (WRVA 2007), November 20–23 2007, Itumbiara, Goiós, Brazil* (pp. 58–61). in Portuguese (original title: Flight Instructor, um Instrutor de Vôo Virtual para o Microsoft Flight Simulator X).

[Decker, 1995] Decker, K. S. (1995). *Environment Centered Analysis and Design of Coordination Mechanisms*. PhD thesis, Department of Computer Science, University of Massachusetts – Amherst.

[Decker, 1996] Decker, K. S. (1996). TÆMS: A Framework for Environment Centered Analysis & Design of Coordination Mechanisms. In G. M. P. O'Hare & N. R. Jennings (Eds.), *Foundations of Distributed Artificial Intelligence*, John Wiley Sixth-Generation Computer Technology Series chapter 16, (pp. 429–448). John Wiley & Sons, Inc.

[DeLoach & Wood, 2001] DeLoach, S. A. & Wood, M. F. (2001). Developing Multiagent Systems with AgentTool. In C. Castelfranchi & Y. Lespérance (Eds.), *Proceedings of the 7th International Workshop on Intelligent Agents: Agent Theories, Architectures and Languages (ATAL'00), July 7–9 2000, Boston, Massachusetts, USA*, volume 1986 of *Lecture Notes in Computer Science* (pp. 46–60).: Springer-Verlag.

[DeLoach et al., 2001] DeLoach, S. A., Wood, M. F., & Sparkman, C. H. (2001). Multiagent Systems Engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3), 231–258.

[Desai et al., 1998] Desai, J. P., Ostrowski, J., & Kumar, V. (1998). Controlling Formations of Multiple Mobile Robots. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation, May 16–20 1998, Leuven , Belgium* (pp. 2864–2869).

[Deursen et al., 2007] Deursen, D. V., Bruyne, S. D., Lancker, W. V., Neve, W. D., Schrijver, D. D., Hellwagner, H., & de Walle, R. V. (2007). MuMiVA: a Multimedia Delivery Platform using Format-Agnostic, XML-Driven Content Adaptation. In *Proceedings of the 9th IEEE International Symposium on Multimedia (ISM '07), December 10–12 2007, Taichung, Taiwan* (pp. 131–138).: IEEE Computer Society.

[Dias et al., 2006] Dias, P. S., Goncalves, G. M., Gomes, R. M. F., Sousa, J. B., Pinto, J., & Pereira, F. L. (2006). Mission Planning and Specification in the Neptus Framework. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA 2006), May 15–19 2006, Orlando, Florida, USA* (pp. 3220–3225).

[DiCristofaro & Hanna, 1989] DiCristofaro, D. C. & Hanna, S. R. (1989). *OCD: The Offshore and Coastal Dispersion Model – Volume I: User's Guide*. Technical Report AO85-1, U.S. Environmental Protection Agency.

[Diehl et al., 2009] Diehl, A., Abbate, H., Delrieux, C., & Gambini, J. (2009). Integration of Flight Simulators and Geographic Databases. *CLEI Electronic Journal*, 12(3), 8 pages.

[DOD, 2005] DOD (2005). *Unmanned Aircraft Systems Roadmap 2005-2030*. Technical report, Department of Defense. Office of the Secretary of Defense.

[Dogan et al., 2005] Dogan, A., Sato, S., & Blake, W. (2005). Flight Control and Simulation for Aerial Refueling. In *Proceedings of the 2005 AIAA Guidance, Navigation, and Control Conference and Exhibit (AIAA 2005), August 15–18 2005, San Francisco, California, USA* (pp. 15 pages).

[Duarte et al., 2005] Duarte, C. N., Martel, G. R., Buzzell, C., Crimmins, D., Komerska, R., Mupparapu, S., Chappell, S., Blidberg, D. R., & Nitzel, R. (2005). A Common Control Language to Support Multiple Cooperating AUVs. In *Proceedings of the 14th International Symposium on Unmanned Untethered Submersible Technology (UUST 2005), August 21–24 2005, Durham, New Hampshire, USA* (pp. 9 pages).

[Duarte et al., 2004] Duarte, C. N., Martel, G. R., Eberbach, E., & Buzzell, C. (2004). Talk Amongst Yourselves: Getting Multiple Autonomous Vehicles to Cooperate. In *Proceedings of the 2004 IEEE/OES Autonomous Underwater Vehicles, June 17–18 2004* (pp. 96–101).

[Duarte & Werger, 2000] Duarte, C. N. & Werger, B. B. (2000). Defining a Common Control Language for Multiple Autonomous Vehicle Operation. In *Proceedings of the 2000 MTS/IEEE Conference and Exhibition (OCEANS 2000), September 11–14 2000, Providence, Rhode Island, USA* (pp. 1861–1867).

[Durfee, 1999] Durfee, E. H. (1999). Distributed Problem Solving and Planning. In G. Weiß (Ed.), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence* (pp. 121–164). MIT Press.

[Durfee & Lesser, 1987] Durfee, E. H. & Lesser, V. R. (1987). Using Partial Global Plans to Coordinate Distributed Problem Solvers. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI '87), August 23–28, 1987, Milan, Italy*, volume 2 (pp. 875–883).: Morgan Kaufmann Publishers Inc.

[Durfee & Rosenschein, 1994] Durfee, E. H. & Rosenschein, J. S. (1994). Distributed Problem Solving and Multiagent Systems: Comparisons and Examples. In M. Klein (Ed.), *Proceedings of the 13th International Workshop on Distributed Artificial Intelligence (DAI), Lake Quinalt, Washington, USA, July 1994* (pp. 94–104).

[Endo et al., 2006] Endo, Y., Ali, K. S., Balch, T. R., Cameron, J. M., Chen, Z., Halliburton, W. C., Kaess, M., Kira, Z., Lee, J. B., MacKenzie, D. C., Martinson, E. B., Merrill, E. P., Ranganathan, A., Sgorbissa, A., Stoytchev, A., Ulam, P., & Wagner, A. (2006). *MissionLab User Manual for MissionLab version 7.0*. Georgia Tech Mobile Robot Laboratory, College of Computing, Georgia Institute of Technology, Atlanta, Georgia, USA.

[Erdos & Watkins, 2008] Erdos, D. & Watkins, S. (2008). UAV Autopilot Integration and Testing. In *Proceedings of the 2008 IEEE Region 5 Conference, April 17–20 2008, Kansas City, Missouri, USA* (pp. 94–99).: IEEE.

[Esposito et al., 2007] Esposito, F., Rufino, G., Moccia, A., Donnarumma, P., Esposito, M., & Magliulo, V. (2007). An integrated electro-optical payload system for forest fires monitoring from airborne platform. In *2007 IEEE Aerospace Conference, 3–10 March 2007, Big Sky, Montana, USA* (pp. 1–13).

[FAA, 2010a] FAA (2010a). *Air Traffic Organization Policy*. Order JO 7110.65T, Federal Aviation Administration (FAA), U.S. Department of Transportation.

[FAA, 2010b] FAA (2010b). Temporary Flight Restrictions. Federal Aviation Administration. Available online at `http://tfr.faa.gov/tfr2/list.html` (accessed January 2011).

[Fan et al., 2009] Fan, H., Meng, L., & Jahnke, M. (2009). Generalization of 3D Buildings Modelled by CityGML. In M. Sester, L. Bernard, & V. Paelke (Eds.), *Advances in GIScience, Proceedings of the 12th AGILE Conference, Hannover, Germany, 2–5 June 2009*, Lecture Notes in Geoinformation and Cartography (pp. 387–405).: Springer Berlin Heidelberg.

[Fan et al., 2006] Fan, X., Yen, J., Miller, M. S., Ioerger, T. R., & Volz, R. A. (2006). MALLET – A Multi-Agent Logic Language for Encoding Teamwork. *IEEE Transactions on Knowledge and Data Engineering*, 18(1), 123–138.

[Fan et al., 2005] Fan, X., Yen, J., Miller, M. S., & Volz, R. A. (2005). The Semantics of MALLET – An Agent Teamwork Encoding Language. In J. A. Leite, A. Omicini, P. Torroni, & P. Yolum (Eds.), *Second International Workshop on Declarative Agent Languages and Technologies II (DALT 2004), July 19 2004, New York, NY, USA, Revised Selected Papers*, volume 3476 of *Lecture Notes in Computer Science* (pp. 69–91).: Springer-Verlag Berlin Heidelberg.

[Faruqi & Vu, 2002] Faruqi, F. A. & Vu, T. L. (2002). *Mathematical Models for a Missile Autopilot Design*. Technical Note DSTO-TN-0449, Defense Science and Technology Organisation (DSTO) Systems Sciences Laboratory, Australia.

[Ferber & Drogoul, 1992] Ferber, J. & Drogoul, A. (1992). Using Reactive Multi-Agent Systems in Simulation and Problem Solving. In N. M. Avouris & L. G. Gasser (Eds.), *Distributed Artificial Intelligence: Theory and Praxis* (pp. 53–80). Kluwer Academic Publishers.

[Finin et al., 1994] Finin, T., Weber, J., Wiederhold, G., Genesereth, M. R., Fritzson, R., McKay, D., Shapiro, S. C., McGuire, J., Pelavin, R., & Beck, C. (1994). *Specification of the KQML Agent-Communication Language plus Example Agent Policies and Architectures*. Technical report, The DARPA Knowledge Sharing Initiative External Interfaces Working Group.

[FIPA, 1999] FIPA (1999). *Extending UML for the Specification of Agent Interaction Protcols*. Technical Report ad/99-12-03, Foundation for Intelligent Physical Agents (FIPA) Technical Committee C: Agent Communication, Geneve, Switzerland.

[FIPA, 2001] FIPA (2001). *FIPA Ontology Service Specification*. Experimental Component XC00086D, Foundation for Intelligent Physical Agents (FIPA), Geneve, Switzerland.

[FIPA, 2002a] FIPA (2002a). *FIPA ACL Message Structure Specification*. Standard SC00061G, Foundation for Intelligent Physical Agents (FIPA), Geneve, Switzerland.

[FIPA, 2002b] FIPA (2002b). *FIPA Contract Net Interaction Protocol Specification*. Standard SC00029H, Foundation for Intelligent Physical Agents (FIPA), Geneve, Switzerland.

[FIPA, 2002] FIPA (2002). *FIPA Contract Net Interaction Protocol Specification*. Standard Component SC00029H, Foundation for Intelligent Physical Agents (FIPA), Geneve, Switzerland.

[FIPA, 2004] FIPA (2004). *FIPA Agent Management Specification*. Standard Component SC00023K, Foundation for Intelligent Physical Agents (FIPA), Geneve, Switzerland.

[Flight One Software, Inc., 2009] Flight One Software, Inc. (2009). *Airport Facilitator X Manual*. Flight One Software, Inc., 1.08 edition.

[Fong & Thorpe, 2001] Fong, T. & Thorpe, C. (2001). Vehicle Teleoperation Interfaces. *Autonomous Robots*, 11(1), 9–18.

[Franklin & Graesser, 1996] Franklin, S. & Graesser, A. C. (1996). Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents. In J. P. Müller, M. Wooldridge, & N. R. Jennings (Eds.), *Proceedings of the Third International Workshop on Intelligent Agents, Agent Theories, Architectures and Languages (ATAL) (held with ECAI '96), Budapest, Hungary, August 12–13 1996*, volume 1193 of *Lecture Notes in Computer Science* (pp. 21–35).: Springer-Verlag.

[Frazzoli et al., 2000] Frazzoli, E., Dahleh, M. A., & Feron, E. (2000). Robust Hybrid Control for Autonomous Vehicle Motion Planning. In *Proceedings of the 39th IEEE Conference on Decision and Control, December 12–15 2000, Sydney, Australia* (pp. 821–826).

[Friedman-Hill, 2003] Friedman-Hill, E. (2003). *Jess in Action: Java Rule-Based Systems*. In Action. Manning Publications.

[Frost, 1996] Frost, J. R. (1996). *The Theory of Search: A Simplified Explanation*. Technical Report 96-F-HNG040, Soza & Company, Ltd. and Office of Search and Rescue, U.S. Coast Guard.

[Future Combat Systems, 2008] Future Combat Systems (2008). Future Combat Systems (Brigade Combat Team) (FCS (BCT)) System Overview. White paper, US Army. Available online at `https://www.fcs.army.mil/news/pdf/FCS_White_Paper_APR08.pdf` (accessed July 2010).

[García-Ojeda et al., 2006] García-Ojeda, J. C., Arenas, A. E., & de Jesús Pérez-Alcázar, J. (2006). Paving the Way for Implementing Multiagent Systems: Integrating Gaia with Agent-UML. In J. P. Muller & F. Zambonelli (Eds.), *Proceedings of the 6th International Workshop on Agent-Oriented Software Engineering (AOSE 2005), Utrecht, The Netherlands, 25-26 July 2005* (pp. 179–189).: Springer-Verlag.

[Garcia & Barnes, 2010] Garcia, R. & Barnes, L. (2010). Multi-UAV Simulator Utilizing X-Plane. *Journal of Intelligent & Robotic Systems, Special Issue: Selected papers from the 2nd International Symposium on UAVs, June 8–10 2009, Reno, Nevada, USA*, 57(1–4), 393–406.

[Garro & Turci, 2003] Garro, A. & Turci, P. (2003). *Meta-Model Sources: Gaia*. Technical report, Foundation for Intelligent Physical Agents.

[Gasser et al., 1987] Gasser, L. G., Braganza, C., & Herman, N. (1987). MACE: A Flexible Testbed for Distributed AI Research. In M. N. Huhns (Ed.), *Distributed Artificial Intelligence* (pp. 119–152). Pitman Publishers.

[Genesereth et al., 1992] Genesereth, M. R., Fikes, R. E., Bobrow, D., Brachman, R., Gruber, T., Hayes, P., Letsinger, R., Lifschitz, V., MacGregor, R., McCarthy, J., Norvig, P., Patil, R., & Schubert, L. (1992). *Knowledge Interchange Format Version 3.0 Reference Manual*. Report Logic-92-1, The DARPA Knowledge Sharing Effort Interlingua Working Group.

[Genesereth et al., 1986] Genesereth, M. R., Ginsberg, M. L., & Rosenschein, J. S. (1986). Co-operation without Communication. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI '86), August 11–15 1986, Philadelphia, Pennsylvania, USA*, volume 1 (pp. 51–57).: Morgan Kaufmann.

[Genesereth & Ketchpel, 1994] Genesereth, M. R. & Ketchpel, S. P. (1994). Software Agents. *Communications of the ACM*, 37(7), 48–53.

[Georgieva & Georgiev, 2010] Georgieva, A. & Georgiev, B. (2010). Nontraditional Approach to XML Web Services Interactions. In *Proceedings of the Fifth International Conference on Internet and Web Applications and Services, May 9–15 2010, Barcelona, Spain* (pp. 67–72).: IEEE Computer Society.

[Gerkey et al., 2003] Gerkey, B. P., Vaughan, R. T., & Howard, A. (2003). The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In U. Nunes, A. T. de Almeida, A. K. Bejczy, K. Kosuge, & J. T. Machado (Eds.), *Proceedings of the 11th International Conference on Advanced Robotics (ICAR 2003), June 30 – July 3 2003, Coimbra, Portugal* (pp. 317–323).

[Ghafoor et al., 2004] Ghafoor, A., ur Rehman, M., Khan, Z. A., Ali, A., Ahmad, H. F., & Suguri, H. (2004). SAGE: Next Generation Multi-Agent System. In H. R. Arabnia (Ed.), *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '04), June 21–24 2004, Las Vegas, Nevada, USA*, volume 1 (pp. 139–145).: CSREA Press.

[Gimenes et al., 2008] Gimenes, R., Silva, D. C., Reis, L. P., & Oliveira, E. (2008). Flight Simulation Environments Applied to Agent-Based Autonomous UAVs. In J. Cordeiro & J. Filipe (Eds.), *Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS 2008), June 12–16 2008, Barcelona, Spain*, volume Software Agents and Internet Computing (SAIC) (pp. 243–246).

[Giorgini et al., 2004] Giorgini, P., Kolp, M., Mylopoulos, J., & Pistore, M. (2004). The Tropos Methodology: An Overview. In F. Bergenti, M.-P. Gleizes, & F. Zambonelli (Eds.), *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, Multiagent Systems, Artificial Societies, and Simulated Organizations chapter 5, (pp. 89–106). Kluwer Academic Publishing, 1st edition.

[Giunchiglia et al., 2003] Giunchiglia, F., Mylopoulos, J., & Perini, A. (2003). The Tropos Software Development Methodology: Processes, Models and Diagrams. In F. Giunchiglia, J. Odell, & G. Weiß (Eds.), *Proceedings of the 3rd International Workshop on Agent-Oriented Software Engineering (AOSE 2002), Revised Papers and Invited Contributions, July 15 2002, Bologna, Italy*, volume 2585 of *Lecture Notes in Computer Science* (pp. 162–173).: Springer.

[GlobalSecurity.org, 2005a] GlobalSecurity.org (2005a). Multifunction utility/logistics equipment vehicle (mule) ugv. Available online at `http://www.globalsecurity.org/military/systems/ground/fcs-mule.htm` (accessed May 2011).

[GlobalSecurity.org, 2005b] GlobalSecurity.org (2005b). TALON Small Mobile Robot. Available online at `http://www.globalsecurity.org/military/systems/ground/talon.htm` (accessed May 2011).

[Goldsman et al., 2009] Goldsman, D., Nance, R. E., & Wilson, J. R. (2009). A Brief History of Simulation. In M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, & R. G. Ingalls (Eds.), *Proceedings of the 2009 Winter Simulation Conference (WSC 2009), December 13–16 2009, Austin, Texas, USA* (pp. 310–313).

[Gonzalez-Palacios & Luck, 2008] Gonzalez-Palacios, J. & Luck, M. (2008). Extending Gaia with Agent Design and Iterative Development. In M. Luck & L. Padgham (Eds.), *Proceedings of the 8th International Workshop on Agent-Oriented Software Engineering (AOSE 2007), Revised Selected Papers, May 14 2007, Honolulu, Hawaii, USA*, volume 4951 of *Lecture Notes in Computer Science* (pp. 16–30).: Springer.

[Goodrich et al., 2008] Goodrich, M. A., Morse, B. S., Gerhardt, D., Cooper, J. L., Quigley, M., Adams, J. A., & Humphrey, C. (2008). Supporting Wilderness Search and Rescue using a

Camera-Equipped Mini UAV. *Journal of Field Robotics Special Issue on Search and Rescue Robots*, 25(1–2), 89–110.

[Goodrick, 2000] Goodrick, T. (2000). Flight Dynamics for Microsoft Flight Simulator.

[Google, 2010] Google (2010). Google Elevation API. Google Maps Documentation. Available online at `http://code.google.com/apis/maps/documentation/elevation/` (accessed July 2010).

[Gorodetsky et al., 2007] Gorodetsky, V., Karsaev, O., Kupin, V., & Samoilov, V. (2007). Agent-Based Air Traffic Control in Airport Airspace. In *Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'07), November 2–5 2007, Silicon Valley, CA, USA* (pp. 81–84).: IEEE Computer Society.

[Gottuk et al., 2002] Gottuk, D. T., Peatross, M. J., Roby, R. J., & Beyler, C. L. (2002). Advanced Fire Detection Using Multi-Signature Alarm Algorithms. *Fire Safety Journal*, 37(4), 381 – 394.

[Grassle et al., 1979] Grassle, J., Berg, C., Childress, J., Hessler, R., Jannasch, H., Karl, D., Lutz, R., Mickel, T., Rhoads, D., Sanders, H., Smith, K., Somero, G., Turner, R., Tuttle, J., Walsh, P., & Williams, A. (1979). Galápagos '79: Initial Findings of a Deep-Sea Biological Quest. *Oceanus*, 22(2), 2–10.

[Grosso et al., 2003] Grosso, A., Coccoli, M., Gozzi, A., & Boccalatte, A. (2003). An Agent Programming Framework Based on the C# Language and the CLI. In *Proceedings of the 1st International Workshop on C# and .NET Technologies on Algorithms, Computer Graphics, Visualization, Distributed and WEB Computing (C# and .Net Technologies 2003), February 5–7 2003, Plzen, Czech Republic* (pp. 13–20).

[Grupping, 2008] Grupping, J. (2008). Flight Simulator History. Web Site. Available online at `http://fshistory.simflight.com/fsh/index.htm` (accessed September 2010).

[Gutknecht & Ferber, 2001] Gutknecht, O. & Ferber, J. (2001). The MADKIT Agent Platform Architecture. In T. Wagner & O. F. Rana (Eds.), *Proceedings of the International Workshop on Infrastructure for Multi-Agent Systems: Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems, Revised Papers, June 3–7 2000, Barcelona, Spain*, volume 1887 of *Lecture Notes in Computer Science* (pp. 48–55).: Springer Berlin / Heidelberg.

[Gutknecht et al., 2000] Gutknecht, O., Ferber, J., & Michel, F. (2000). *The MADKIT Agent Platform Architecture*. Rapport de Recherche R.R.LIRMM 000xx, Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, France.

[Hallenborg, 2008] Hallenborg, K. (2008). *Advanced Studies in Software and Knowledge Engineering*, volume 4 of *Information Science & Computing*, chapter Core Design Pattern for Efficient Multi-agent Architecture, (pp. 29–36). Institute of Information Theories and Applications FOI ITHEA, Sofia, 1000, P.O.B. 775, Bulgaria. Supplement to the International Journal "Information Technologies & Knowledge", Volume 2 / 2008.

[Hargrove et al., 2000] Hargrove, W. W., Gardner, R., Turner, M., Romme, W., & Despain, D. (2000). Simulating Fire Patterns in Heterogeneous Landscapes. *Ecological Modelling*, 135(2-3), 243–263.

[Haulman, 2003] Haulman, D. L. (2003). *U.S. Unmanned Aerial Vehicles in Combat, 1991-2003*. Technical Report A330434, Air Force Historical Research Agency, Maxwell Air Force Base, Alabama, USA.

[Hayes-Roth, 1995] Hayes-Roth, B. (1995). An Architecture for Adaptive Intelligent Systems. *Artificial Intelligence*, 72(1–2), 329–365.

[Hermans & Decuypere, 2005] Hermans, D. & Decuypere, R. (2005). A Challenge for Micro and Mini UAV: The Sensor Problem. In *Proceedings of the Meeting in Advanced Sensory Payloads for UAV* (pp. 8 pages).

[Hexmoor & Heng, 2000] Hexmoor, H. & Heng, T. (2000). Air Traffic Control and Alert Agent. In C. Sierra, M. Gini, & J. S. Rosenschein (Eds.), *Proceedings of the fourth international conference on Autonomous agents (AGENTS'00), June 3–7 2000, Barcelona, Spain* (pp. 237–238).: ACM.

[Hoagland et al., 2010] Hoagland, P., Beaulieu, S., Tivey, M. A., Eggert, R. G., German, C., Glowka, L., & Lin, J. (2010). Deep-Sea Mining of Seafloor Massive Sulfides. *Marine Policy*, 34(3), 728–732.

[Hobbs, 2003] Hobbs, R. L. (2003). Using XML to Support Military Decision-Making. In L. Wood (Ed.), *Proceedings of the XML Conference and Exposition 2003 (XML 2003), December 7–12 2003, Philadelphia, Pennsylvania, USA*.

[Hoffmann et al., 1999] Hoffmann, F., Koo, T.-J., & Shakernia, O. (1999). Evolutionary Design of a Helicopter Autopilot. In *Advances in Soft Computing – Engineering Design and Manufacturing, Part 3: Intelligent Control* (pp. 201–214).: Springer-Verlag.

[Huang et al., 2009] Huang, C.-H., Chuang, T.-R., Deng, D.-P., & Lee, H.-M. (2009). Building GML-Native Web-Based Geographic Information Systems. *Computers & Geosciences*, 35(9), 1802–1816.

[Huhns & Stephens, 1999] Huhns, M. N. & Stephens, L. M. (1999). Multiagent Systems and Societies of Agents. In G. Weiß (Ed.), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence* (pp. 79–120). MIT Press.

[Hur et al., 2003] Hur, Y., Fierro, R. B., & Lee, I. (2003). Modeling Distributed Autonomous Robots using CHARON: Formation Control Case Study. In *Proceedings of the 6th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'03), May 14–16 2003, Hakodate, Hokkaido, Japan* (pp. 93–96).: IEEE Computer Society.

[Iglesias et al., 1999] Iglesias, C. A., Garijo, M., & Gonzalez, J. C. (1999). A Survey of Agent-Oriented Methodologies. In J. P. Müller, M. P. Singh, & A. S. Rao (Eds.), *Proceedings of the 5th International Workshop on Intelligent Agents: Agent Theories, Architectures and Languages (ATAL'98), July 4–7 1998, Paris, France*, volume 1555 of *Lecture Notes in Computer Science* (pp. 317–330).: Springer-Verlag.

[Jackson, 1995] Jackson, E. B. (1995). *Manual for a Workstation-Based Generic Flight Simulation Program (LaRCSim)*. Technical Memorandum NASA-TM-110164, NASA Langley Research Center, Virginia, USA.

[Jackson, 2007] Jackson, J. (2007). Microsoft Robotics Studio: A Technical Introduction. *Robotics & Automation Magazine*, 14(4), 82–87.

[Jackson & Robins, 1994] Jackson, M. & Robins, I. (1994). Gas Sensing for Fire Detection: Measurements of CO, CO2, H2, O2, and Smoke Density in European Standard Fire Tests. *Fire Safety Journal*, 22(2), 181–205.

[Jäger & Nebel, 2002] Jäger, M. & Nebel, B. (2002). Dynamic Decentralized Area Partitioning for Cooperating Cleaning Robots. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation (ICRA '02), May 11–15 2002, Washington, DC, USA* (pp. 3577–3582).: IEEE.

[Jager, 2008] Jager, R. W. (2008). Test and Evaluation of the Piccolo II Autopilot System on a One-Third Scale Yak-54. Master's thesis, University of Kansas.

[Jakuba, 2007] Jakuba, M. V. (2007). *Stochastic Mapping for Chemical Plume Source Localization with Application to Autonomous Hydrothermal Vent Discovery*. PhD thesis, Massachusetts Institute of Technology and Woods Hole Oceanographic Institution.

[jen Chiang et al., 1997] jen Chiang, Y., Klosowski, J. T., Lee, C., & Mitchell, J. S. B. (1997). Geometric Algorithms for Conflict Detection/Resolution in Air Traffic Management. In *Proceedings of the 36th IEEE Conference on Decision and Control, December 10–12 1997, San Diego, CA, USA* (pp. 1835–1840).: IEEE.

[Jennings, 1996] Jennings, N. R. (1996). Coordination Techniques for Distributed Artificial Intelligence. In G. M. P. O'Hare & N. R. Jennings (Eds.), *Foundations of Distributed Artificial Intelligence*, John Wiley Sixth-Generation Computer Technology Series (pp. 187–210). John Wiley & Sons, Inc.

[Johnson, 2008] Johnson, C. W. (2008). Using Evacuation Simulations for Contingency Planning to Enhance the Security and Safety of the 2012 Olympic Venues. *Safety Science*, 46(2), 302–322.

[Juan et al., 2002] Juan, T., Pearce, A., & Sterling, L. (2002). Roadmap: Extending the Gaia Methodology for Complex Open Systems. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Aystems (AAMAS '02), July 15–19 2002, Bologna, Italy* (pp. 3–10).: ACM.

[Kahan et al., 2000] Kahan, M., Tanzer, J., Darvin, D., & Borer, F. (2000). Virtual Reality-Assisted Cognitive-Behavioral Treatment for Fear of Flying: Acute Treatment and Follow-up. *CyberPsychology & Behavior*, 3(3), 387–392.

[Kanchanavally, 2006] Kanchanavally, S. (2006). *Cooperative Navigation of Autonomous Vehicles in a Known Environment*. PhD thesis, School of Engineering, University of Dayton, Ohio.

[Katevas, 2001] Katevas, N. I., Ed. (2001). *Mobile Robotics in Health Care Services*, volume 7 of *Assistive Technology Research*. IOS Press, 1st edition.

[Kelley et al., 2001] Kelley, D. S., Karson, J. A., Blackman, D. K., Früh-Green, G. L., Butterfield, D. A., Lilley, M. D., Olson, E. J., Schrenk, M. O., Roe, K. K., Lebon, G. T., Rivizzigno, P., & the AT3-60 Shipboard Party (2001). An Off-Axis Hydrothermal Vent Field Near the Mid-Atlantic Ridge at 30^{o}N. *Nature*, 412, 145–149.

[Kelley et al., 2005] Kelley, D. S., Karson, J. A., Früh-Green, G. L., Yoerger, D. R., Shank, T. M., Butterfield, D. A., Hayes, J. M., Schrenk, M. O., Olson, E. J., Proskurowski, G., Jakuba, M. V., Bradley, A. M., Larson, B., Ludwig, K., Glickson, D., Buckman, K., Bradley, A. S., Brazelton, W. J., Roe, K., Elend, M. J., Delacour, A., Bernasconi, S. M., Lilley, M. D., Baross, J. A., Summons, R. E., & Sylva, S. P. (2005). A Serpentinite-Hosted Ecosystem: The Lost City Hydrothermal Field. *Science*, 307(5714), 1428–1434.

[Kenny et al., 2008] Kenny, J., Takeda, K., & Thomas, G. (2008). Real-time Computational Fluid Dynamics for Flight Simulation. In *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC), December 1–4 2008, Orlando, Florida, USA*.

[Khalique et al., 2007] Khalique, S., Farooq, S., Ahmad, H. F., Suguri, H., & Ali, A. (2007). SAGE-LITE: An Architecture and Implementation of Light Weight Multiagent System. In *Proceedings of the 8th International Symposium on Autonomous Decentralized Systems (ISADS'07), March 21–23 2007, Sedona, Arizona, USA* (pp. 239–244).: IEEE Computer Society.

[Kim et al., 2009] Kim, J.-H., Jeong, I.-B., Park, I.-W., & Lee, K.-H. (2009). Multi-Layer Architecture of Ubiquitous Robot System for Integrated Services. *International Journal of Social Robotics*, 1(1), 19–28.

[Kitano et al., 1997] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., & Osawa, E. (1997). Robocup: The Robot World Cup Initiative. In *Proceedings of the First International Conference on Autonomous Agents, Marina del Rey, California, USA, February 05–08 1997* (pp. 340–347).: ACM.

[Kroll, 2008] Kroll, A. (2008). A Survey on Mobile Robots for Industrial Inspections. In W. Bulgard, R. Dillmann, C. Plagemann, & N. Vahrenkamp (Eds.), *Proceedings of the 10th International Conference on Intelligent Autonomous Systems (IAS 2008), July 23–25 2008, Baden Baden, Germany* (pp. 406–414).: IOS Press.

[Krose et al., 2004] Krose, B., Bunschoten, R., Hagen, S., Terwijn, B., & Vlassis, N. (2004). Household Robots Look and Learn: Environment Modeling and Localization from an Omnidirectional Vision System. *IEEE Robotics & Automation Magazine*, 11(4), 45–52.

[Krozel & Peters, 1997] Krozel, J. & Peters, M. (1997). Strategic Conflict Detection and Resolution for Free Flight. In *Proceedings of the 36th IEEE Conference on Decision and Control, December 10–12 1997, San Diego, CA, USA* (pp. 1822–1828).: IEEE.

[Krozel et al., 2001] Krozel, J., Peters, M., Bilimoria, K. D., Lee, C., , & Mitchell, J. S. (2001). System Performance Characteristics of Centralized and Decentralized Air Traffic Separation Strategies. In *Proceedings of the Fourth USA/Europe Air Traffic Management Research and Development Seminar, December 3–7 2001, Santa Fe, NM, USA*.

[Kumar et al., 2009] Kumar, C. S., Govardhan, A., & Rao, C. G. (2009). Usage of XML Technology in Electronic Health Record for Effective Heterogeneous Systems Integration in Healthcare. *International Journal of Medical Engineering and Informatics*, 1(4), 399–406.

[Labrou & Finin, 1997] Labrou, Y. & Finin, T. (1997). *A Proposal for a new KQML Specification*. Technical Report CS-97-03, University of Maryland Baltimore County - Computer Science and Electrical Engineering Department.

[Lancaster, 2004] Lancaster, R. (2004). Formation Flight Autopilot Design for the GAF Jindivik Mk 4A UAV. Master's thesis, Cranfield University – College of Aeronautics.

[Lau & Reis, 2007] Lau, N. & Reis, L. P. (2007). *Robotic Soccer*, chapter FC Portugal – High-Level Coordination Methodologies in Soccer Robotics, (pp. 167–192). I-Tech Education and Publishing.

[Lee et al., 2003] Lee, J., Huang, R., Vaughn, A., Xiao, X., Hedrick, J. K., Zennaro, M., & Sengupta, R. (2003). Strategies of Path-Planning for a UAV to Track a Ground Vehicle. In *Proceedings of the 2nd annual Autonomous Intelligent Networks and Systems Conference, June 30–July 1 2003, Menlo Park, California, USA* (pp. 6 pages).

[Lee et al., 1998] Lee, L. C., Nwana, H. S., Ndumu, D. T., & Wilde, P. D. (1998). The Stability, Scalability and Performance of Multi-Agent Systems. *BT Technology Journal*, 16(3), 94–103.

[Lesser, 1999] Lesser, V. R. (1999). Cooperative Multiagent Systems: A Personal View of the State of the Art. *IEEE Transactions on Knowledge and Data Engineering*, 11(1), 133–142.

[Lesser & Corkill, 1981] Lesser, V. R. & Corkill, D. D. (1981). Functionally Accurate Cooperative Distributed Systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1), 81–96.

[Lesser & Corkill, 1987] Lesser, V. R. & Corkill, D. D. (1987). Distributed Problem Solving. In S. C. Shapiro & D. Eckroth (Eds.), *Encyclopedia of Artificial Intelligence*, volume 1 (pp. 245–251). John Wiley & Sons.

[Leszczyna, 2008] Leszczyna, R. (2008). *Evaluation of Agent Platforms (ver. 2.0)*. Technical Report JRC 47224, Institute for the Protection and Security of the Citizen, European Commission's Joint Research Centre, Italy.

[Lewis & Jacobson, 2002] Lewis, M. & Jacobson, J. (2002). Game Engines in Scientific Research – Introduction. *Communications of the ACM, Special Issue: Game Engines in Scientific Research*, 45(1), 27–31.

[LIDO, 2007] LIDO (2007). *The Ontology Service*. L.I.D.O. (Laboratorio di Informatica Sistemi Distribuiti e ad Oggetti), Genova, Italy, 1st edition.

[LIDO, 2008a] LIDO (2008a). *AgentService Mobile: A Light Architecture for Mobile Devices*. L.I.D.O. (Laboratorio di Informatica Sistemi Distribuiti e ad Oggetti), Genova, Italy, 1st edition.

[LIDO, 2008b] LIDO (2008b). *The Ontology Agent*. L.I.D.O. (Laboratorio di Informatica Sistemi Distribuiti e ad Oggetti), Genova, Italy, 1st edition.

[LIDO, 2009a] LIDO (2009a). *AgentService External Runtime*. L.I.D.O. (Laboratorio di Informatica Sistemi Distribuiti e ad Oggetti), Genova, Italy, 1st edition.

[LIDO, 2009b] LIDO (2009b). *Federation Management Suite*. L.I.D.O. (Laboratorio di Informatica Sistemi Distribuiti e ad Oggetti), Genova, Italy, 1st edition.

[MacArthur, 2006] MacArthur, E. Z. (2006). *Compliant Formation Control of an Autonomous Multiple Vehicle System*. PhD thesis, University of Florida.

[Macedonia, 2002] Macedonia, M. (2002). Games, Simulation, and the Military Education Dilemma. Publications from the Forum for the Future of Higher Education.

[Mackenzie & Arkin, 1998] Mackenzie, D. C. & Arkin, R. C. (1998). Evaluating the Usability of Robot Programming Toolsets. *The International Journal of Robotics Research*, 17(4), 381–401.

[MacKenzie et al., 1997] MacKenzie, D. C., Arkin, R. C., & Cameron, J. M. (1997). Multiagent Mission Specification and Execution. *Autonomous Robots*, 4(1), 29–52.

[Maes, 1996] Maes, P. (1996). Intelligent Software: Easing the Burdens that Computers Put on People. *IEEE Expert: Intelligent Systems and Their Applications*, 11(6), 62–63.

[Malheiro & Oliveira, 2000] Malheiro, B. & Oliveira, E. (2000). Solving Conflicting Beliefs with a Distributed Belief Revision Approach. In M. C. Monard & J. S. Sichman (Eds.), *Proceedings of the International Joint Conference 7th Ibero-American Conference on Artificial Intelligence 15th Brazilian Symposium on Artificial Intelligence IBERAMIA-SBIA 2000, November 19–22, 2000, Atibaia, São Paulo, Brazil*, volume 1952 of *Lecture Notes in Computer Science* (pp. 146–155).: Springer Berlin / Heidelberg.

[Mallah et al., 2009] Mallah, G. A., Shaikh, Z. A., & Shaikh, N. A. (2009). Towards an Agent Framework to Support Distributed Problem Solving. In N. Mastorakis, V. Mladenov, & V. T. Kontargyri (Eds.), *Proceedings of the European Computing Conference, Volume 1, September 25–27 2007, Athens, Greece*, volume 27 of *Lecture Notes Electrical Engineering* (pp. 329–335).: Springer US.

[Malone & Crowston, 1994] Malone, T. W. & Crowston, K. (1994). The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, 26(1), 87–119.

[Marshall, 2005] Marshall, J. A. (2005). *Coordinated Autonomy: Pursuit Formations of Multi-vehicle Systems*. PhD thesis, Graduate Department of Electrical and Computer Engineering, University of Toronto.

[Martínez-de-Dios et al., 2007] Martínez-de-Dios, J. R., Merino, L., Ollero, A., Ribeiro, L. M., & Viegas, X. (2007). Multi-uav experiments: Application to forest fires. In A. Ollero & I. Maza (Eds.), *Multiple Heterogeneous Unmanned Aerial Vehicles*, volume 37 of *Springer Tracts in Advanced Robotics* chapter 8, (pp. 207–228). Springer Berlin / Heidelberg.

[Mashyanov et al., 2001] Mashyanov, N., Sholupov, S., Ryzhov, V., Pogarev, S., Ilyin, T., Reshetov, V., Palinkaš, L., Durn, G., Miko, S., Špiric, Z., Dragaš, M., Zvonaric, T., Pirc, S., Gosar, M., Bidovec, M., Sajn, R., & Gruntorad, J. (2001). Detection of pollution sources using continuous mercury automobile survey. In *6th International Conference on Mercury as a Global Pollutant (6 ICMGP) : Book of Abstracts*.

[Masterson et al., 2009] Masterson, J., Keeshan, B., & Hauck, H. (2009). *Airport Design Editor User Manual (English Version)*. The ScruffyDuck Software Company, 1.47 edition.

[McCarley & Wickens, 2005] McCarley, J. S. & Wickens, C. D. (2005). *Human Factors Implications of UAVs in the National Airspace*. Technical Report AHFD-05-05/FAA-05-01, Aviation Human Factors Division of the Institute of Aviation, University of Illinois at Urbana-Champaign.

[Microsoft Corporation, 2008a] Microsoft Corporation (2008a). Compiling BGL. Microsoft Developer Network. Part of Microsoft ESP SDK. Available online at `http://msdn.microsoft.com/en-us/library/cc526978.aspx` (accessed July 2010).

[Microsoft Corporation, 2008b] Microsoft Corporation (2008b). ESP SDK Overview. Microsoft Developer Network Library. Available online from `http://msdn.microsoft.com/en-us/library/cc526948.aspx` (accessed September 2010).

[Microsoft Corporation, 2010] Microsoft Corporation (2010). XML Schema Definition Tool (Xsd.exe). Microsoft Developer Network Library. Available online at `http://msdn.microsoft.com/en-us/library/x6c1kb0s.aspx` (accessed September 2010).

[Miller, 2006] Miller, B. D. (2006). Improvised Explosive Devise Placement Detection from a Semi-Autonomous Ground Vehicle. Master's thesis, Naval Postgraduate School, Monterey, California, USA.

[Miller et al., 2000] Miller, M. S., Yin, J., Volz, R. A., Ioerger, T. R., & Yen, J. (2000). Training Teams with Collaborative Agents. In G. Gauthier, C. Frasson, & K. VanLehn (Eds.), *Proceedings of the 5th International Conference on Intelligent Tutoring Systems (ITS '00), June 19–23 2000, Montréal, Canada*, volume 1839 of *Lecture Notes in Computer Science* (pp. 63–72).: Springer-Verlag.

[Mindell & Bingham, 2001] Mindell, D. & Bingham, B. (2001). New Archaeological uses of Autonomous Underwater Vehicles. In *Proceedings of the 2001 MTS/IEEE OCEANS Conference and Exhibition, November 5–8 2001, Honolulu, Hawaii, USA* (pp. 555–558).

[Mooney, 1997] Mooney, C. Z. (1997). *Monte Carlo Simulation*, volume 07-116 of *Sage University Papers series on Quantitative Applications in the Social Sciences*. Sage Publications.

[Moraitis & Spanoudakis, 2006] Moraitis, P. & Spanoudakis, N. I. (2006). The Gaia2Jade Process for Multi-Agent Systems Development. *Applied Artificial Intelligence*, 20(2-4), 251–273.

[Morgan, 2008] Morgan, S. (2008). *Programming Microsoft Robotics Studio*. Microsoft Press, 1st edition.

[Mota et al., 2011] Mota, L., Reis, L. P., & Lau, N. (2011). Multi-Robot Coordination using Setplays in the Middle-Size and Simulation Leagues. *Mechatronics*, 21(2), 434–444.

[Moulin & Chaib-draa, 1996] Moulin, B. & Chaib-draa, B. (1996). An Overview of Distributed Artificial Intelligence. In G. M. P. O'Hare & N. R. Jennings (Eds.), *Foundations of Distributed Artificial Intelligence*, John Wiley Sixth-Generation Computer Technology Series (pp. 3–55). John Wiley & Sons, Inc.

[Mueller & Fischer, 1995] Mueller, H. C. & Fischer, A. (1995). Robust Fire Detection Algorithm for Temperature and Optical Smoke Density Using Fuzzy Logic. In L. D. Sanson (Ed.), *Proceedings of the 29th Annual 1995 International Carnahan Conference on Security Technology; Sanderstead; UK; 18-20 October 1995* (pp. 197–204).

[Myers, 2008] Myers, K. (2008). Fatigue Mitigation Initiatives in the FAA's Air Traffic Organization. Presented at the FAA Fatigue Management Symposium: Partnerships for Solutions; June 17-19, 2008, Vienna, Virginia, USA.

[Naden, 1971] Naden, R. A. (1971). *Detection of Anomalous Air Pollution Sources and Guidelines for the Design of Source Surveillance Systems*. PhD thesis, Rice University.

[Nance & Sargent, 2002] Nance, R. E. & Sargent, R. G. (2002). Perspectives on the Evolution of Simulation. *Operations Research*, 50(1), 161–172.

[NASA, 2008] NASA (2008). Vertical Motion Simulator. Available online at `http://www.aviationsystemsdivision.arc.nasa.gov/facilities/vms/index.shtml` (accessed January 2009).

[Nguyen et al., 2002] Nguyen, G., Dang, T., Hluchy, L., Laclavik, M., Balogh, Z., & Budinska, I. (2002). *Agent Platform Evaluation and Comparison*. Technical Report 5FP IST-2001-34519, Institute of Informatics, Slovak Academy of Sciences.

[Nonami, 2007] Nonami, K. (2007). Prospect and Recent Research & Development for Civil Use Autonomous Unmanned Aircraft as UAV and MAV. *Journal of System Design and Dynamics*, 1(2), 120–128.

[NRC, 2005] NRC (2005). *Autonomous Vehicles in Support of Naval Operations*. The National Academies Press. Committee on Autonomous Vehicles in Support of Naval Operations, Naval Studies Board, Division on Engineering and Physical Sciences, National Research Council.

[Nwana, 1996] Nwana, H. S. (1996). Software Agents: An Overview. *Knowledge Engineering Review*, 11(3), 205–244.

[Nwana et al., 1996] Nwana, H. S., Lee, L., & Jennings, N. R. (1996). Coordination in Software Agent Systems. *British Telecom Technical Journal*, 14(4), 79–88.

[Nwana et al., 1998] Nwana, H. S., Ndumu, D. T., & Lee, L. C. (1998). ZEUS: An Advanced Tool-Kit for Engineering Distributed Multi-Agent Systems. In H. Nwana & D. Ndumu (Eds.), *Proceedings of the 3rd International Conference on Practical Applications of Intelligent Agents and Multi-Agent Technology (PAAM'98), London, UK, March 23-25, 1998* (pp. 377–391).: Practical Application Co Ltd.

[Odell et al., 2001] Odell, J., Parunak, H. V. D., & Bauer, B. (2001). Representing Agent Interaction Protocols in UML. In P. Ciancarini & M. Wooldridge (Eds.), *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering (AOSE 2000), Revised Papers, June 10 2000, Limerick, Ireland*, volume 1957 of *Lecture Notes in Computer Science* (pp. 121–140).: Springer-Verlag New York, Inc.

[OGC, 2007a] OGC (2007a). *OpenGIS Geography Markup Language (GML) Encoding Standard*. OpenGIS Standard OGC 07-036, Open Geospatial Consortium Inc.

[OGC, 2007b] OGC (2007b). *OpenGIS Sensor Model Language (SensorML) Implementation Specification*. OpenGIS Implementation Specification OGC 07-000, Open Geospatial Consortium Inc.

[OGC, 2008a] OGC (2008a). *OGC KML*. OpenGIS Implementation Specification OGC 07-147r2, Open Geospatial Consortium Inc.

[OGC, 2008b] OGC (2008b). *OpenGIS City Geography Markup Language (CityGML) Encoding Standard*. OpenGIS Encoding Standard OGC 08-007r1, Open Geospatial Consortium Inc.

[OMG, 2005] OMG (2005). *Software Process Engineering Metamodel Specification*. Specification formal/05-01-06, Object Management Group.

[OMG, 2008] OMG (2008). *Software & Systems Process Engineering Meta-Model Specification*. Specification formal/2008-04-01, Object Management Group.

[Padgham et al., 2008] Padgham, L., Thangarajah, J., & Winikoff, M. (2008). The Prometheus Design Tool - a Conference Management System Case Study. In M. Luck & L. Padgham (Eds.), *Proceedings of the 8th International Workshop on Agent-Oriented Software Engineering (AOSE 2007), Revised Selected Papers, May 14 2007, Honolulu, Hawaii, USA*, volume 4951 of *Lecture Notes in Computer Science* (pp. 197–211).: Springer.

[Padgham & Winikoff, 2003] Padgham, L. & Winikoff, M. (2003). Prometheus: a Methodology for Developing Intelligent Agents. In F. Giunchiglia, J. Odell, & G. Weiß (Eds.), *Proceedings*

of the 3rd International Workshop on Agent-Oriented Software Engineering (AOSE 2002), Revised Papers and Invited Contributions, July 15 2002, Bologna, Italy, volume 2585 of *Lecture Notes in Computer Science* (pp. 174–185).: Springer.

[Padgham & Winikoff, 2004] Padgham, L. & Winikoff, M. (2004). *Developing Intelligent Agent Systems: a Practical Guide*. Wiley Series in Agent Technology. John Wiley and Sons, 1st edition.

[Page, 2000] Page, R. L. (2000). Brief History of Flight Simulation. In *Proceedings of the SimTecT 2000, February 28 – March 2 2000, Sydney, Australia* (pp. 11 pages).

[Parodi et al., 2009] Parodi, O., Lapierre, L., & Jouvencel, B. (2009). Hardware-in-the-Loop Simulators for Multi-Vehicles Scenarios: Survey on Existing Solutions and Proposal of a New Architecture. In *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009), October 11–15 2009, St. Louis, Missouri, USA* (pp. 225–230).

[Peel, 2001] Peel, R. (2001). Airport, Navigation Aid and IFR Intersection Data in FlightGear. FlightGear Development Documents. Available online at `http://www.flightgear.org/Docs/AirNav/AptNavFAQ.FlightGear.html` (accessed July 2010).

[Peel, 2009] Peel, R. (2009). *X-Plane Airport Data (Apt.Dat) File Specification*, 850 edition. Available online at `http://data.x-plane.com/file_specs/XP%20APT850%20Spec.pdf` (accessed July 2010).

[Pellanda et al., 2002] Pellanda, P. C., Apkarian, P., & Tuan, H. D. (2002). Missile Autopilot Design Via a Multi-Channel LFT/LPV Control Method. *International Journal of Robust and Nonlinear Control*, 12(1), 1–20.

[Perry, 2004] Perry, A. R. (2004). The FlightGear Flight Simulator. In *Proceedings of the USENIX 2004 Annual Technical Conference (USENIX '04), June 27 – July 2 2004, Boston, Massachusetts, USA* (pp. 171–182).: USENIX Association.

[Pfister, 1997] Pfister, G. (1997). Multisensor/Multicriteria Fire Detection: A New Trend Rapidly Becomes State of the Art. *Fire Technology*, 33(2), 115–139.

[Poslad et al., 2000] Poslad, S., Buckle, P., & Hadingham, R. (2000). The FIPA-OS Agent Platform: Open Source for Open Standards. In *Proceedings of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 2000), April 10–12 2000, Manchester, UK* (pp. 355–368).: The Practical Application Company.

[Pěchouček et al., 2006] Pěchouček, M., Šišlák, D., Pavlíček, D., & Uller, M. (2006). Autonomous Agents for Air-Traffic Deconfliction. In P. Stone & G. Weiss (Eds.), *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '06), May 8–12 2006, Hakodate, Japan* (pp. 1498–1505).: ACM.

[Rafi et al., 2006] Rafi, F., Khan, S., Shafiq, K., & Shah, M. (2006). Autonomous Target Following by Unmanned Aerial Vehicles. In G. R. Gerhart, C. M. Shoemaker, & D. W. Gage (Eds.), *Proceedings of the SPIE Defense and Security Symposium, May 2006, Orlando, Florida, USA* (pp. 8 pages).

[Rajkovic et al., 2008] Rajkovic, B., Grsic, Z., & Vujadinovic, M. (2008). Detection of a possible source of air pollution using a combination of measurements and inverse modelling. In C. Borrego & A. I. Miranda (Eds.), *Air Pollution Modeling and its Application XIX*, NATO Science for Peace and Security Series C: Environmental Security (pp. 689–690). Springer Netherlands.

[Rango & Nahavandi, 2007] Rango, R. D. & Nahavandi, S. (2007). Simulating Autonomous Robot Teams with Microsoft Robotics Studio. In *Proceedings of the 2007 Simulation Conference and Exhibition (SimTecT 2007), June 4–7 2007, Brisbane, Queensland, Australia* (pp. 6 pages).: Simulation Industry Association of Australia.

[Ranney et al., 2003] Ranney, T. A., Ranney, T. A., Watson, G., Salaani, K., Mazzae, E. N., & Grygier, P. (2003). *Investigation of Driver Reactions to Tread Separation Scenarios in the National Advanced Driving Simulator (NADS)*. Technical Report DOT HS 809 523, US Department of Transportation – National Highway Traffic Safety Administration (NHTSA) – Vehicle Research and Test Center, Ohio, USA.

[Rao & Georgeff, 1991] Rao, A. S. & Georgeff, M. P. (1991). Modeling Rational Agents within a BDI Architecture. In J. Allen, R. Fikes, & E. Sandewall (Eds.), *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR '91)* (pp. 473–484).: Morgan Kaufmann publishers Inc.

[Reis, 2003] Reis, L. P. (2003). *Coordenação em Sistemas Multi-Agente: Aplicações na Gestão Universitária e Futebol Robótico*. PhD thesis, Faculty of Engineering, University of Porto, Porto, Portugal.

[Reis & Lau, 2001a] Reis, L. P. & Lau, N. (2001a). COACH UNILANG – A Standard Language for Coaching a (Robo)Soccer Team. In A. Birk, S. Coradeschi, & S. Tadokoro (Eds.), *Proceedings of RoboCup 2001: Robot Soccer World Cup V, Seattle, Washington, USA*, volume 2377 of *Lecture Notes in Computer Science* (pp. 183–192).: Springer.

[Reis & Lau, 2001b] Reis, L. P. & Lau, N. (2001b). FC Portugal Team Description: RoboCup 2000 Simulation League Champion. In P. H. Stone, T. R. Balch, & G. K. Kraetzschmar (Eds.), *RoboCup 2000: Robot Soccer World Cup IV*, volume 2019 of *Lecture Notes in Computer Science* (pp. 29–40).: Springer Berlin / Heidelberg.

[Reynolds, 1987] Reynolds, C. W. (1987). Flocks, Herds and Schools: A Distributed Behavioral Model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)* (pp. 25–34).: ACM.

[Ribeiro & Oliveira, 2010] Ribeiro, L. R. & Oliveira, N. M. F. (2010). UAV Autopilot Controllers Test Platform Using Matlab/Simulink and X-Plane. In *Proceedings of the 40th ASEE/IEEE Frontiers in Education Conference (FIE 2010), October 27–30 2010, Arlington, Virginia, USA*.

[Rinehart, 1969] Rinehart, J. S. (1969). Old Faithful Geyser Performance: 1870 through 1966. *Bulletin of Volcanology*, 33(1), 153–163.

[RoboCup, 2010] RoboCup (2010). Robocup. Website. Available online at `http://www.robocup.org/` (accessed July 2010).

[Rosen, 2008] Rosen, K. R. (2008). The History of Medical Simulation. *Journal of Critical Care*, 23(2), 157–166.

[Rothbaum et al., 2006] Rothbaum, B. O., Anderson, P., Zimand, E., Hodges, L., Lange, D., & Wilson, J. (2006). Virtual Reality Exposure Therapy and Standard (in vivo) Exposure Therapy in the Treatment for the Fear of Flying. *Behavior Therapy*, 37(1), 80–90.

[Rozinat et al., 2009] Rozinat, A., Wynn, M., van der Aalst, W., ter Hofstede, A., & Fidge, C. (2009). Workflow Simulation for Operational Decision Support. *Data & Knowledge Engineering*, 68(9), 834–850.

[Russel & Norvig, 2002] Russel, S. J. & Norvig, P. (2002). *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, 2nd edition.

[Russell, 1980] Russell, J. A. (1980). A Circumplex Model of Affect. *Journal of Personality and Social Psychology*, 39(6), 1161–1178.

[Rusu et al., 2007] Rusu, R. B., Maldonado, A., Beetz, M., & Gerkey, B. (2007). Extending Player/Stage/Gazebo towards Cognitive Robots Acting in Ubiquitous Sensor-equipped Environments. In *Proceedings of the 2007 IEEE International Conference on Robotics and Automation (ICRA 2007) Workshop for Network Robot System, April 14 2007, Rome, Italy* (pp. 9 pages).

[Ryan et al., 2004] Ryan, A., Zennaro, M., Howell, A., Sengupta, R., & Hedrick, J. K. (2004). An Overview of Emerging Results in Cooperative UAV Control. In *Proceedings of 43rd IEEE Conference on Decision and Control, December 14–17 2004, Paradise Island, Bahamas* (pp. 602–607).

[Rysdyk, 2006] Rysdyk, R. (2006). Unmanned Aerial Vehicle Path Following for Target Observation in Wind. *Journal of Guidance, Ccontrol, and Dynamics*, 29(5), 1092–1100.

[Saigol et al., 2010] Saigol, Z. A., Dearden, R. W., Wyatt, J. L., & Murton, B. J. (2010). Belief Change Maximisation for Hydrothermal Vent Hunting Using Occupancy Grids. In T. Belpaeme, G. Bugmann, C. Melhuish, & M. Witkowski (Eds.), *Proceedings of the 11th Conference Towards Autonomous Robotic Systems (TAROS 2010), August 31 – September 2 2010, Plymouth, UK* (pp. 247–254).

[Santos, 2010] Santos, A. (2010). Autonomous Intelligent Vehicle Adaptation and Performance Analysis in Flight Simulator X. Master's thesis, Faculty of Engineering, University of Porto, Porto, Portugal.

[Sarton, 2003] Sarton, C. J. (2003). Autopilot Using Differential Thrust for ARIES Autonomous Underwater Vehicle. Master's thesis, Naval Postgraduate School, Monterey, California, USA.

[Scheck, 2004] Scheck, W. (2004). Lawrence Sperry: Autopilot Inventor and Aviation Innovator. *Aviation History Magazine*, 2(5).

[Schiff, 1971] Schiff, B. J. (1971). *Flying, A Golden Science Guide*. Golden Press.

[Schneider-Fontán & Matarić, 1998] Schneider-Fontán, M. & Matarić, M. J. (1998). Territorial Multi-Robot Task Division. *IEEE Transactions on Robotics and Automation*, 14(5), 815–822.

[Schulman & Scire, 1980] Schulman, L. L. & Scire, J. S. (1980). *Buoyant Line and Point Source (BLP) Dispersion Model User's Guide*. Technical Report P-7304B, Environmental Research & Technology, Inc.

[Schwarz et al., 2003] Schwarz, C., Gates, T., & Papelis, Y. (2003). Motion Characteristics of the National Advanced Driving Simulator. In *Proceedings of the Driving Simulation Conference North America 2003 (DSC-NA 2003), October 8–10 2003, Dearborn, Michigan, USA* (pp. 15 pages).

[Schweiger et al., 2005] Schweiger, R., Brumhard, M., Hoelzer, S., & Dudeck, J. (2005). Implementing Health Care Systems using XML Standards. *International Journal of Medical Informatics*, 74(2), 267–277.

[Scire et al., 2000] Scire, J. S., Strimaitis, D. G., & Yamartino, R. J. (2000). *A User's Guide for the CALPUFF Dispersion Model*. Earth Tech, Inc., 196 Baker Avenue, Concord, MA 01742, USA, 5 edition.

[Seetharaman et al., 2006] Seetharaman, G., Lakhotia, A., & Blasch, E. P. (2006). Unmanned Vehicles Come of Age: The DARPA Grand Challenge. *Computer*, 39(12), 26–29.

[Sen & Weiß, 1999] Sen, S. & Weiß, G. (1999). Learning in Multiagent Systems. In G. Weiß (Ed.), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence* (pp. 259–298). MIT Press.

[Sholes, 2007] Sholes, E. (2007). Evolution of a UAV Autonomy Classification Taxonomy. In *Proceedings of the 2007 IEEE Aerospace Conference, March 3–10 2007, Big Sky, Montana, USA* (pp. 16 pages).

[Sichman & Demazeau, 1995] Sichman, J. S. & Demazeau, Y. (1995). Exploiting Social Reasoning to Deal with Agency Level Inconsistency. In V. R. Lesser & L. G. Gasser (Eds.), *Proceedings of the 1st International Conference of Multi-Agent Systems, (ICMAS'95), June 12-14 1995, San Francisco, California, USA* (pp. 352–359).: MIT Press.

[Silva, 2011] Silva, D. C. (2011). *Platform User's Manual*. Department of Informatics Engineering, Faculty of Engineering, University of Porto / Artificial Intelligence and Computer Science Laboratory. Available online at `http://wwww.fe.up.pt/~dcs/phd/PlatformUserManual.pdf` (accessed June 2011).

[Silva et al., 2007a] Silva, D. C., Abreu, P., Mendes, P., & Vinhas, V. (2007a). Integrador Automático de Notícias. In J. C. Ramalho, J. C. Lopes, & L. Carriço (Eds.), *Proceedings of the 5th National Conference on XML: Associated Applications and Technologies (XATA 2007), February 15–16 2007, Lisbon, Portugal* (pp. 223–234).

[Silva et al., 2007b] Silva, D. C., Abreu, P., Mendes, P., & Vinhas, V. (2007b). Voice OverM2L – Talk the Math. In J. C. Ramalho, J. C. Lopes, & L. Carriço (Eds.), *Proceedings of the 5th National Conference on XML: Associated Applications and Technologies (XATA 2007), February 15–16 2007, Lisbon, Portugal* (pp. 94–105).

[Silva et al., 2011a] Silva, D. C., Abreu, P., Reis, L. P., & Oliveira, E. (2011a). Development of a Flexible Language for Mission Description for Multi-Robot Missions. *Journal of Intelligent and Robotic System*, Submitted.

[Silva et al., 2010] Silva, D. C., Braga, R. A. M., Reis, L. P., & Oliveira, E. (2010). A Generic Model for a Robotic Agent System using GAIA Methodology: Two Distinct Implementations. In *Proceedings of the 2010 IEEE Conference on Robotics, Automation and Mechatronics (RAM 2010), June 28–30 2010, Singapore* (pp. 280–285).: IEEE.

[Silva et al., 2011b] Silva, D. C., Braga, R. A. M., Reis, L. P., & Oliveira, E. (2011b). Designing a Meta-Model for a Generic Robotic Agent System using Gaia Methodology. *Information Sciences*, Accepted for publication.

[Silva et al., 2005] Silva, D. C., Moreira, V. V., & Pereira, T. F. (2005). ERS-210 Mobile Video Surveillance System. In L. P. Reis, C. Carreto, E. Silva, & N. Lau (Eds.), *1st International Workshop on Intelligent Robotics (IROBOT'05), held with the 12th Portuguese Conference on Artificial Intelligence (EPIA 2005), December 2–5 2005, Covilhã, Portugal* (pp. 262–265).

[Silva et al., 2011c] Silva, D. C., Reis, L. P., & Oliveira, E. (2011c). Development of a Flexible Language for Disturbance Description for Multi-Robot Missions. *Computational Intelligence*, Submitted.

[Silva et al., 2011d] Silva, D. C., Reis, L. P., & Oliveira, E. (2011d). Development of Flexible Languages for Scenario and Team Description in Multi-Robot Missions. *Computer Standards & Interfaces*, Submitted.

[Silva et al., 2009a] Silva, D. C., Silva, R., Reis, L. P., & Oliveira, E. (2009a). Agent-Based Aircraft Control Strategies in a Simulated Environment. In F. Dignum, J. M. Bradshaw, B. G. Silverman, & W. A. van Doesburg (Eds.), *Agents for Games and Simulations – Trends in Techniques, Concepts and Design (Proceedings of the 1st International Workshop on Agents for Games and Simulations (AGS 2009), May 11 2009, Budapest, Hungary)*, volume 5920 of *Lecture Notes in Computer Science* (pp. 190–205).: Springer Berlin / Heidelberg.

[Silva et al., 2009b] Silva, D. C., Vinhas, V., Reis, L. P., & Oliveira, E. (2009b). Biometric Emotion Assessment and Feedback in an Immersive Digital Environment. *International Journal of Social Robotics*, 1(4), 307–317.

[Skogestad, 2003] Skogestad, S. (2003). Simple Analytic Rules for Model Reduction and PID Controller Tuning. *Journal of Process Control*, 13(4), 291–309.

[Smith et al., 1994] Smith, D. C., Cypher, A., & Spohrer, J. (1994). KidSim: Programming Agents Without a Programming Language. *Communications of the ACM*, 37(7), 54–67.

[Smith & Davis, 1981] Smith, R. G. & Davis, R. (1981). Frameworks for Cooperation in Distributed Problem Solving. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1), 61–70.

[Sorton & Hammaker, 2005] Sorton, E. F. & Hammaker, S. (2005). Simulated Flight Testing of an Autonomous Unmanned Aerial Vehicle Using FlightGear. In *Proceedings of the AIAA Infotech@Aerospace 2005 Workshop, Conference and Exhibit, September 26–29 2005, Arlington, Virginia, USA* (pp. 13 pages).: American Institute of Aeronautics and Astronautics.

[Sousa, 2010] Sousa, P. D. (2010). Autonomous Air Traffic Control for Intelligent Vehicles using Microsoft Flight Simulator X. Master's thesis, Faculty of Engineering, University of Porto, Porto, Portugal.

[Sousa et al., 2010] Sousa, P. D., Silva, D. C., & Reis, L. P. (2010). Air Traffic Control with Microsoft Flight Simulator X. In Álvaro Rocha, C. F. Sexto, L. P. Reis, & M. P. Cota (Eds.), *Actas de la 5ª Conferencia Ibérica de Sistemas y Tecnologías de Información (CISTI 2010), June 16–19 2010, Santiago de Compostela, Spain* (pp. 378–383). in Portuguese (original title: Controlo de Tráfego Aéreo com o Microsoft Flight Simulator X).

[Stall & Bourne, 1996] Stall, D. A. & Bourne, S. (1996). The National Advanced Driving Simulator: Potential Applications to ITS and AHS Research. In *Proceedings of the 1996 Annual Meetings of ITS America, April 1996* (pp. 15 pages).

[Stelzer & Pröll, 2008] Stelzer, R. & Pröll, T. (2008). Autonomous Sailboat Navigation for Short Course Racing. *Robotics and Autonomous Systems*, 56(7), 604–614.

[Stephens et al., 2000] Stephens, G. L., Ellingson, R. G., Jr., J. V., Bolton, W., Tooman, T. P., Valero, F. P. J., Minnis, P., Pilewskie, P., Phipps, G. S., Sekelsky, S., Carswell, J. R., Miller, S. D., Benedetti, A., McCoy, R. B., Jr., R. F. M., Lederbuhr, A., & Bambha, R. (2000). The Department of Energy's Atmospheric Radiation Measurement (ARM) Unmanned Aerospace Vehicle (UAV) Program. *Bulletin of the American Meteorological Society*, 81(12), 2915–2937.

[Sterling & Taveter, 2009] Sterling, L. & Taveter, K. (2009). *The Art of Agent-Oriented Modeling*. Intelligent Robotics and Autonomous Agents. The MIT Press, 1st edition.

[Stewart, 1966] Stewart, D. (1966). A Platform with Six Degrees of Freedom. *Aircraft Engineering and Aerospace Technology*, 38(4), 30–35.

[Stock, 2007] Stock, C. (2007). Flight Dynamics: FSX and X-Plane Battle it out. Sim Pilot Magazine, March 2007. Available online at `http://www.simpilotnet.com/index.php?option=com_content&task=view&id=20` (accessed September 2010).

[Stolarik, 2007] Stolarik, B. (2007). *Intelligent Flight Control Simulation Research Program*. Technical Report AFRL-VA-WP-TR-2007-3054, Air Force Research Laboratory Wright Site (AFRL/WS), Ohio, USA.

[Stoller, 2006] Stoller, G. (2006). Fear of Flying can Cripple Workers. USA Today. Available online at `http://www.usatoday.com/educate/college/business/articles/20060326.htm` (accessed September 2010).

[Stone, 2000] Stone, P. H. (2000). *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*. The MIT Press.

[Stone & Veloso, 1998] Stone, P. H. & Veloso, M. M. (1998). Task Decomposition and Dynamic Role Assignment for Real-Time Strategic Teamwork. In J. P. Müller, M. P. Singh, & A. S. Rao (Eds.), *Proceedings of the 5th International Workshop on Intelligent Agents, Agent Theories, Architectures, and Languages (ATAL '98), July 4–7 1998, Paris, France*, volume 1555 of *Lecture Notes In Computer Science* (pp. 293–308).: Springer-Verlag.

[Stone & Veloso, 2000] Stone, P. H. & Veloso, M. M. (2000). Multiagent Systems: A Survey from a Machine Learning Perspective. *Autonomous Robots*, 8(3), 345–383.

[Sturm & Shehory, 2004] Sturm, A. & Shehory, O. (2004). A Comparative Evaluation of Agent-Oriented Methodologies. In F. Bergenti, M.-P. Gleizes, & F. Zambonelli (Eds.), *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, Multiagent Systems, Artificial Societies, and Simulated Organizations chapter 7, (pp. 127–149). Kluwer Academic Publishing, 1st edition.

[Subbotina et al., 2008] Subbotina, M. M., Thomson, R. E., & Anisimov, M. V. (2008). Numerical Simulation of a Hydrophysical Structure near the Seafloor in a Hydrothermal Venting Region. *Doklady Earth Sciences*, 423A(9), 1423–1426. Published in Russian in Doklady Akademii Nauk 423(3) pp. 401–404, 2008.

[Sutton & Craven, 1998] Sutton, R. & Craven, P. J. (1998). The ANFIS Approach Applied to AUV Autopilot Design. *Neural Computing & Applications*, 7(2), 131–140.

[Swartout et al., 2005] Swartout, W., Gratch, J., Hill, R., Hovy, E., Lindheim, R., Marsella, S., Rickel, J., & Traum, D. (2005). Simulation Meets Hollywood: Integrating Graphics, Sound, Story and Character for Immersive Simulation. In O. Stock & M. Zancanaro (Eds.), *Multimodal Intelligent Information Presentation*, volume 27 of *Text, Speech and Language Technology* (pp. 279–304). Springer, 1st edition.

[Tambe, 1997] Tambe, M. (1997). Towards Flexible Teamwork. *Journal of Artificial Intelligence Research*, 7(1), 83–124.

[Thomson et al., 2005] Thomson, R. E., Subbotina, M. M., & Anisimov, M. V. (2005). Numerical Simulation of Hydrothermal Vent-Induced Circulation at Endeavour Ridge. *Journal of Geophysical Research-Oceans*, 110(1), 14 pages.

[Thrun et al., 2006] Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C. M., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G. R., Davies, B., Ettinger, S., Kaehler, A., Nefian, A. V., & Mahoney, P. (2006). Stanley: The Robot that Won the DARPA Grand Challenge. *Journal of Field Robotics*, 23(9), 661–692.

[Toyota Motor Corporation (TMC), 2007] Toyota Motor Corporation (TMC) (2007). Toyota Develops World-class Driving Simulator. TMC News Release. Available online at `http://www.toyota.co.jp/en/news/07/1126_1.html` (accessed June 2010).

[Tran & Hernandez, 2004] Tran, D. & Hernandez, E. (2004). Use of the Vertical Motion Simulator in Support of the American Airlines Flight 587 Accident Investigation. In *American Institute of Aeronautics and Astronautics Modeling and Simulation Technologies Conference and Exhibit, August 2004, Providence, Rhode Island, USA*.

[Turner, 1994] Turner, D. B. (1994). *Workbook of Atmospheric Dispersion Estimates: An Introduction to Dispersion Modeling*. CRC-Press, 2nd edition.

[Urmson et al., 2007] Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Dolan, J., Duggins, D., Ferguson, D., Galatali, T., Geyer, C., Gittleman, M., Harbaugh, S., Hebert, M., Howard, T., Kelly, A., Kohanbash, D., Likhachev, M., Miller, N., Peterson, K., Rajkumar, R., Rybski, P., Salesky, B., Scherer, S., Woo-Seo, Y., Simmons, R., Singh, S., Snider, J., Stentz, A., Whittaker, W., Ziglar, J., Bae, H., Litkouhi, B., Nickolaou, J., Sadekar, V., Zeng, S., Struble, J., Taylor, M., & Darms, M. (2007). *Tartan Racing: A Multi-Modal Approach to the DARPA Urban Challenge*. DARPA Urban Challenge Semifinalist Technical Papers Tartan Racing.

[USCG, 2006] USCG (2006). Search and Rescue – A Guide for Boat Coxswains. United States Coast Guard.

[Vaglienti et al., 2007] Vaglienti, B., Niculescu, M., & Becker, J. (2007). *HIL/SIL Simulator for the Piccolo Avionics*. CloudCap Technology, Hood River, Oregon, USA.

[van Amerongen & van Nauta Lemke, 1978] van Amerongen, J. & van Nauta Lemke, H. (1978). Optimum Steering of Ships with an Adaptive Autopilot. In *Proceedings of the 5th Ship Control Systems Symposium, October 30 – November 3 1978, Annapolis, Maryland, USA*.

[van Andel & Ballard, 1979] van Andel, T. H. & Ballard, R. D. (1979). The Galápagos Rift at 86°W: Volcanism, Structure, and Evolution of the Rift Valley. *Journal of Geophysical Research*, 84(810), 5390–5406.

[Vecchiola et al., 2003] Vecchiola, C., Coccoli, M., & Boccalatte, A. (2003). Agent Programming Extensions Relying on a Component-based Platform. In W. W. Smari & A. M. Memon (Eds.), *Proceedings of the 2003 IEEE International Conference on Information Reuse and Integration (IRI 2003), October 27–29 2003, Las Vegas, Nevada, USA* (pp. 24–31).: IEEE Systems, Man, and Cybernetics Society (SMC).

[Vecchiola et al., 2008] Vecchiola, C., Grosso, A., & Boccalatte, A. (2008). AgentService: a Framework to Develop Distributed Multiagent Systems. *International Journal of Agent-Oriented Software Engineering*, 2(3), 290–323.

[Vestri et al., 2005] Vestri, C., Bougnoux, S., Bendahan, R., Fintzel, K., Wybo, S., Abad, F., & Kakinami, T. (2005). Evaluation of a Vision-Based Parking Assistance System. In *Proceedings of the 8th International IEEE Conference on Intelligent Transportation Systems, September 13–16 2005, Vienna, Austria* (pp. 131–135).

[Vinhas, 2010] Vinhas, V. (2010). *Realtime Dynamic Multimedia Storyline Based on Online Audience Biometric Information*. PhD thesis, Department of Informatics Engineering, Faculty of Engineering, University of Porto, Rua Dr. Roberto Frias, s/n 4200-465 Porto, Portugal.

[Vinhas et al., 2007] Vinhas, V., Silva, D. C., Abreu, P., & Mendes, P. (2007). A Domain-Specific Flexible Optimization Approach for e-Supply Chain Msanagement. In S. Krishnamurthy & P. Isaías (Eds.), *Proceedings of the IADIS International Conference on e-Commerce 2007, December 7–9 2007, Algarve, Portugal* (pp. 35–42).

[von Alt et al., 2001] von Alt, C., Allen, B., Austin, T., Forrester, N., Goldsborough, R., & amd Roger Stokey, M. P. (2001). Hunting for Mines with REMUS: a High Performance, Affordable, Free Swimming Underwater Robot. In *Proceedings of the 2001 MTS/IEEE OCEANS Conference and Exhibition, November 5–8 2001, Honolulu, Hawaii, USA* (pp. 117–122).

[W3C, 2004] W3C (2004). *XML Schema Part 0: Primer (Second Edition)*. W3C Recommendation REC-xmlschema-0-20041028, World Wide Web Consortium.

[W3C, 2008] W3C (2008). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C Recommendation REC-xml-20081126, World Wide Web Consortium.

[W3C, 2009] W3C (2009). *Mathematical Markup Language (MathML) Version 3.0*. W3C Candidate Recommendation CR-MathML3-20091215, World Wide Web Consortium.

[Wang & Balakirsky, 2008] Wang, J. & Balakirsky, S. (2008). *USARSim: A Game-Based Simulation of Mobile Robots*, 3.1.3 edition.

[Wang, 1989] Wang, P. K. C. (1989). Navigation Strategies for Multiple Autonomous Mobile Robots Moving in Formation. In *IEEE/RSJ International Workshop on Intelligent Robots and Systems '89, September 4–6 1989, Tsukuba, Japan* (pp. 486–493).

[Web3D Consortium, 2008] Web3D Consortium (2008). *Extensible 3D (X3D)*. ISO Standard ISO/IEC 19775-1:2008, Web3D Consortium, Inc.

[Webber & Gomaa, 2004] Webber, D. L. & Gomaa, H. (2004). Modeling Variability in Software Product Lines with the Variation Point Model. *Science of Computer Programming*, 53(3), 305–331.

[Weimer et al., 2009] Weimer, J., Sinopoli, B., & Krogh, B. (2009). Multiple source detection and localization in advection-diffusion processes using wireless sensor networks. In T. P. Baker (Ed.), *Proceedings of the 30th IEEE International Real-Time Systems Symposium (RTSS 2009), Washington, DC, USA, 1-4 December 2009* (pp. 333–342).: IEEE Computer Society.

[Weiß, 1996] Weiß, G. (1996). Adaptation and Learning in Multi-Agent Systems: Some Remarks and a Bibliography. In G. Weiß & S. Sen (Eds.), *Proceedings of the Workshop on Adaptation and Learning in Multi-Agent Systems (held with IJCAI'95), Montréal, Canada, August 21, 1995*, volume 1042 of *Lecture Notes in Computer Science* (pp. 1–21).: Springer Berlin / Heidelberg.

[Whitcomb, 2000] Whitcomb, L. L. (2000). Underwater Robotics: Out of the Research Laboratory and Into the Field. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation (ICRA '00), April 24–28 2000, San Francisco, California, USA* (pp. 709–716).

[Winikoff & Padgham, 2004] Winikoff, M. & Padgham, L. (2004). The Prometheus Methodology. In F. Bergenti, M.-P. Gleizes, & F. Zambonelli (Eds.), *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, Multiagent Systems, Artificial Societies, and Simulated Organizations chapter 11, (pp. 217–234). Kluwer Academic Publishing, 1st edition.

[Wittman Jr., 2009] Wittman Jr., R. L. (2009). Defining a Standard: The Military Scenario Definition Language Version 1.0 Standard. In G. A. Wainer, C. A. Shaffer, R. M. McGraw, & M. J. Chinni (Eds.), *Proceedings of the 2009 Spring Simulation Multiconference (SpringSim 2009), March 22–27 2009, San Diego, California, USA*: SCS/ACM.

[Wood & DeLoach, 2001] Wood, M. F. & DeLoach, S. A. (2001). An Overview of the Multiagent Systems Engineering Methodology. In P. Ciancarini & M. Wooldridge (Eds.), *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering (AOSE 2000), Revised Papers, June 10 2000, Limerick, Ireland*, volume 1957 of *Lecture Notes in Computer Science* (pp. 207–221).: Springer-Verlag New York, Inc.

[Wooldridge et al., 2000] Wooldridge, M., Jennings, N. R., & Kinny, D. (2000). The Gaia Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3), 285–312.

[Wooldridge, 1994] Wooldridge, M. J. (1994). Coherent Social Action. In A. G. Cohn (Ed.), *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI-94), Amsterdam, The Netherlands, August 8–12, 1994* (pp. 279–283).: John Wiley & Sons.

[Wooldridge, 2002] Wooldridge, M. J. (2002). *An Introduction to Multi Agent Systems*. John Wiley & Sons, Ltd.

[Wooldridge & Jennings, 1994] Wooldridge, M. J. & Jennings, N. R. (1994). Agent Theories, Architectures, and Languages: A Survey. In M. J. Wooldridge & N. R. Jennings (Eds.), *Proceedings of the Workshop on Intelligent Agents, Agent Theories, Architectures, and Languages (ATAL) (held with ECAI '94), Amsterdam, The Netherlands, August 8–9, 1994*, volume 890 of *Lecture Notes in Computer Science* (pp. 1–39).: Springer Berlin / Heidelberg.

[Wooldridge & Jennings, 1995] Wooldridge, M. J. & Jennings, N. R. (1995). Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10(2), 115–152.

[Yin et al., 2000] Yin, J., Miller, M. S., Ioerger, T. R., Yen, J., & Volz, R. A. (2000). A Knowledge-Based Approach for Designing Intelligent Team Training Systems. In *Proceedings of the 4th International Conference on Autonomous Agents (AGENTS 2000), June 3–7, 2000, Barcelona, Spain* (pp. 427–434).: ACM.

[Yoerger et al., 2002] Yoerger, D. R., Collier, R., & Bradley, A. M. (2002). Hydrothermal Vent Plume Discovery and Survey with an Autonomous Underwater Vehicle. *American Geophysical Union Fall Meeting Abstracts*.

[Yongqiang & Hearn, 2008] Yongqiang, Z. & Hearn, G. E. (2008). Ship Intelligent Autopilot in Narrow Water. In *Proceedings of the 27th Chinese Control Conference (CCC 2008), July 16–18 2008, Kunming, Yunnan, China* (pp. 243–248).

[Zambonelli et al., 2003] Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2003). Developing Mmultiagent Systems: The Gaia Methodology. *ACM Transactions on Software Engineering Methodologies*, 12(3), 317–370.

Appendix A

Gaia SPEM Model

This appendix presents the Software Process Engineering Metamodel (SPEM) model for an extended version of the Gaia methodology. SPEM is a notation used to describe a concrete software development process. Previous work on this specific field includes a model for the original version of Gaia [Garro & Turci, 2003]. This model was produced by the FIPA[1] Methodology Technical Committee using the previous version of SPEM, version 1.1 [OMG, 2005]. Other works include [García-Ojeda et al., 2006] (the authors use SPEM to describe the integration of Gaia with AUML) and [Moraitis & Spanoudakis, 2006] (SPEM is used to describe the integration of Gaia with the JADE platform[2]). In this work, we use SPEM version 2.0 ([OMG, 2008]) and the modeled version of Gaia was based on Gaia v.2 (as described in section 2.1.7), plus the roles and interaction diagram as proposed in [Castro & Oliveira, 2008]. We present only the higher-level models of the Gaia methodology, but not the more detailed diagrams. For those diagrams, refer to [Garro & Turci, 2003].

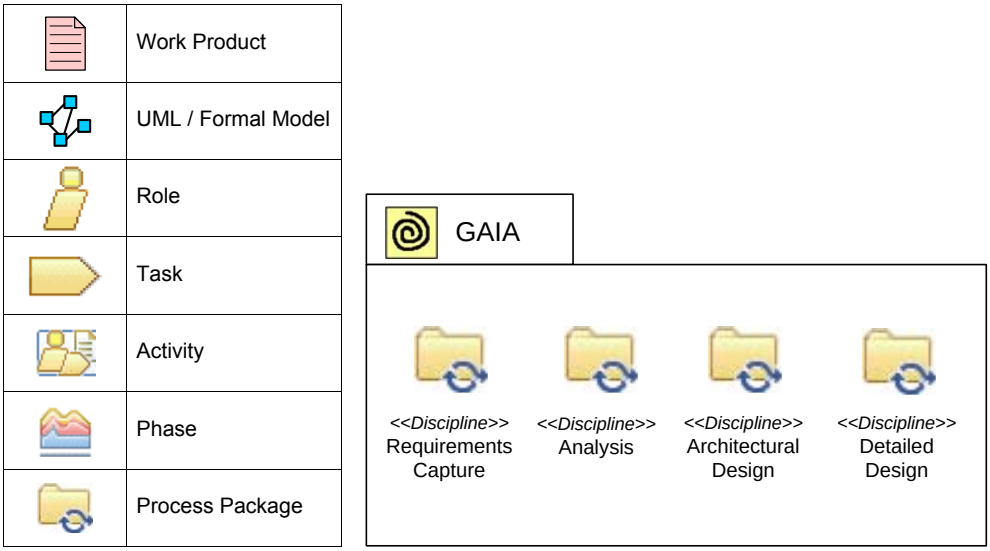
Figure A.1(a) shows a list of some of the stereotypes as defined by SPEM 2.0. Even though the 'UML / formal model' is not defined by the SPEM 2.0 specification, it was included as a legacy stereotype – we considered that it would be helpful to have this stereotype present, to help identify work products with a formal presentation (either UML diagrams or other formally structured models).

The Gaia methodology is here divided into four stages – Requirements Gathering, Analysis, Architectural Design and Detailed Design (see Fig. A.1(b)). Even though the first stage is not actually part of the Gaia methodology, it is present to formally introduce the requirements statement document (Fig. A.2(a)), which is the basis for the remaining stages.

The second stage (analysis) has the objective of developing an understanding of the system and its structure, and a total of five work products (models that can be expressed using schemata templates or informal textual descriptions) are produced – see Fig. A.2(b). The architectural design stage is intended to transform the analysis models into a level of abstraction sufficiently low so that traditional design techniques may be applied in order to implement agents, and involves four work

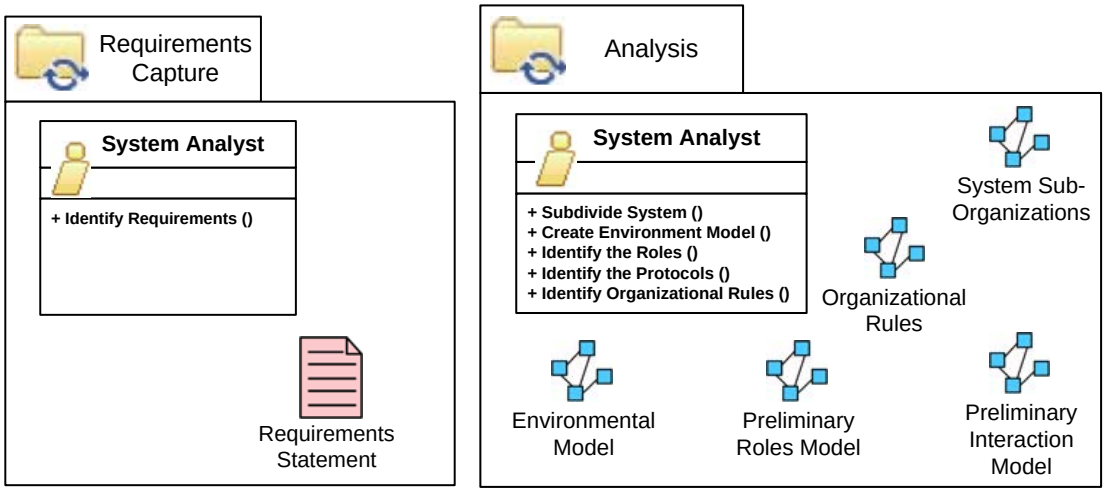
[1] Foundation for Intelligent Physical Agents – see `http://www.fipa.org/` for more information.

[2] The Java Agent DEvelopment framework. More information available at `http://jade.tilab.com/`



(a) SPEM Notation (b) Stages of the Gaia Methodology

Figure A.1: SPEM Notation and Stages of the Gaia Methodology



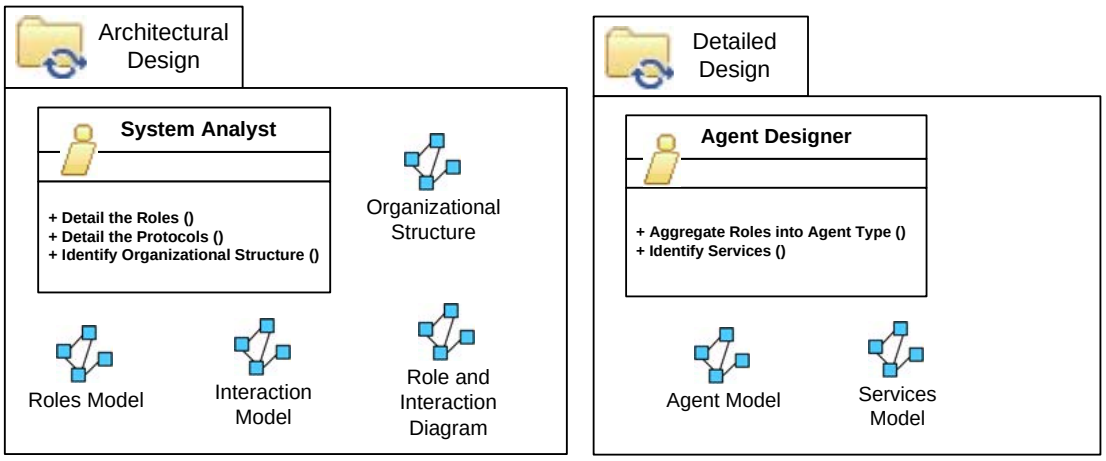
(a) Requirements Package (b) Analysis Package

Figure A.2: Requirements and Analysis Packages

products (including the roles and interaction diagram, as proposed by [Castro & Oliveira, 2008]) – see Fig. A.3(a). Finally, the detailed design stage involves two work products (Fig. A.3(b)): agent model and service model.

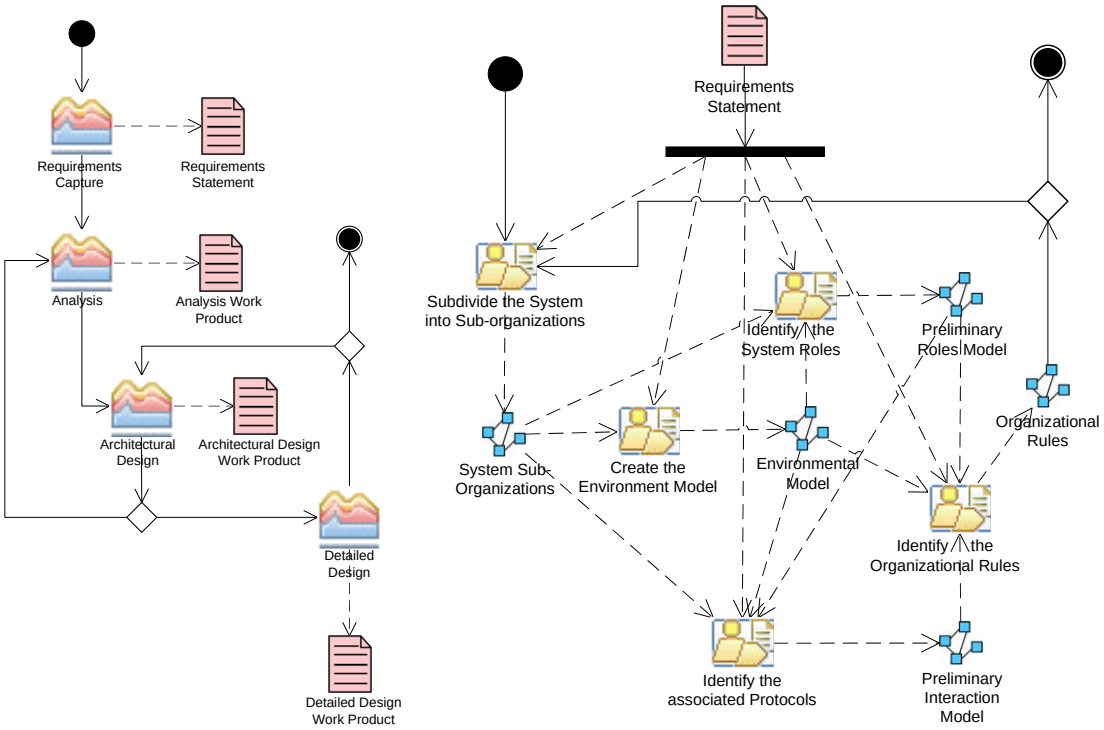
The entire process is described in Fig. A.4(a), which depicts all stages of the Gaia process, and the group of documents produced at each stage.

The analysis stage identifies the sub-organizations present in the system, produces an environment model, a preliminary version of both roles and interaction (patterns of interaction between different roles) models, and the organizational rules (Fig. A.4(b)).



(a) Architectural Design Packages (b) Detailed Design Packages

Figure A.3: Architectural and Detailed Design Packages



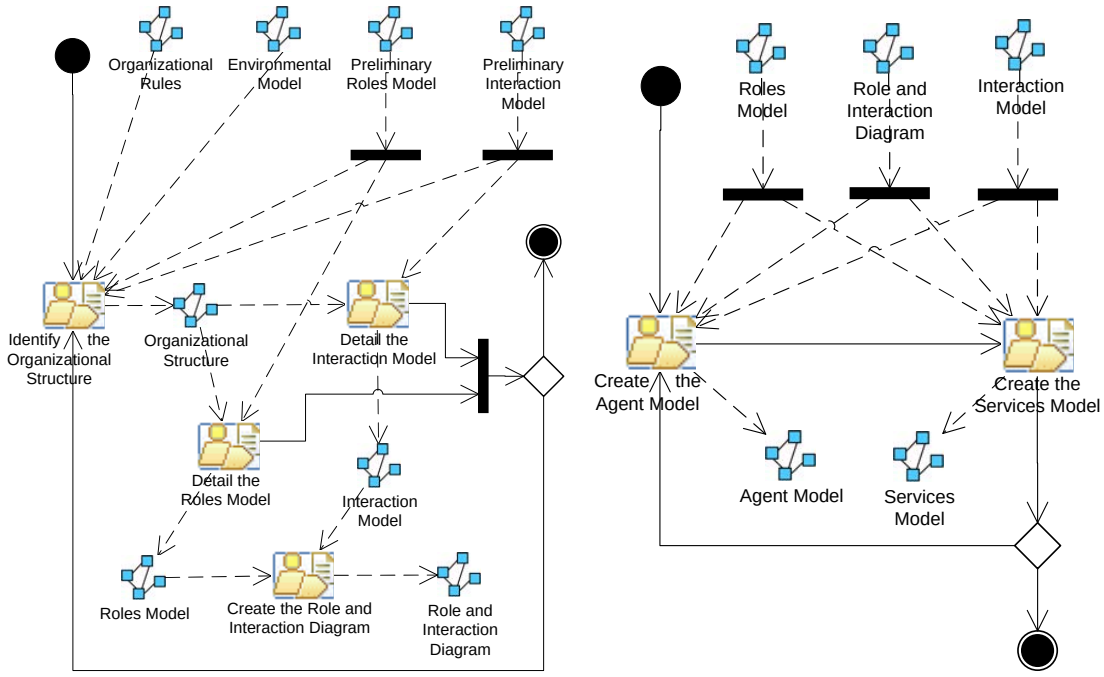
(a) Gaia Process (b) Analysis Stage

Figure A.4: Gaia Process and Analysis Stage

The architectural design stage identifies the organizational structure, details both the roles and the interaction models, and creates the role and interaction diagram – see Fig. A.5(a).

The detailed design stage involves generating two models: the agent model and the services model. The agent model identifies the agent types that will make up the system, and the agent instances that will be instantiated from these types. The services model identifies the main services

that are required to realize the agent's role (Fig. A.5(b)).



(a) Architectural Design Stage (b) Detailed Design Stage

Figure A.5: Architectural and Detailed Design Stages

Appendix B

Dialect Schemas

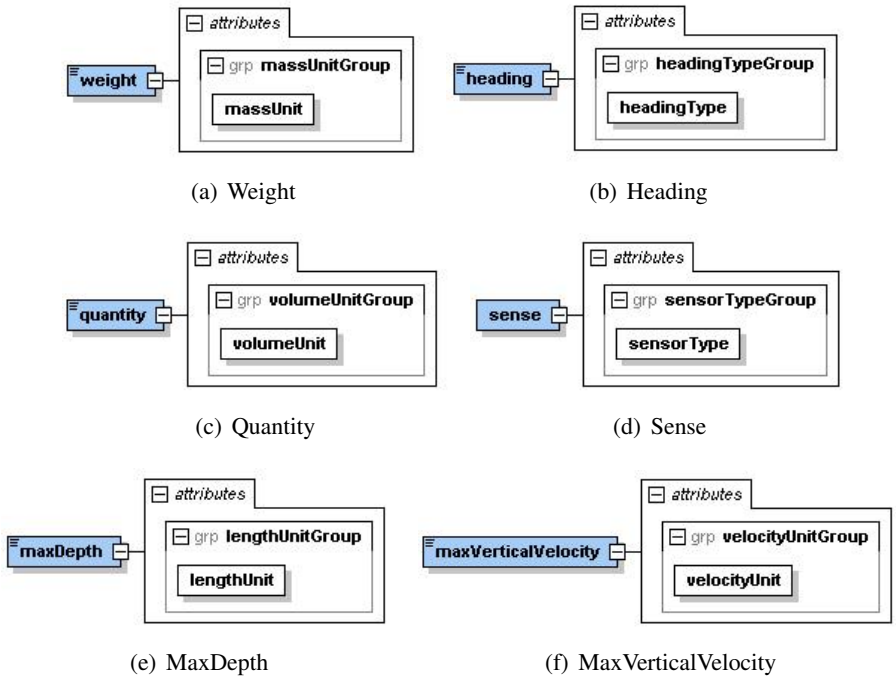
This appendix contains the reference for the developed dialects (SDL, TDL, DDL and MDL), presenting a visual representation of the XML Schema for each dialect[1]. In order to handle elements common to two or more of the dialects (as explained in section 4.1.1), common elements were placed in a file imported by the main schema file of each dialect – common.xsd.

B.1 Common Elements

This file contains several elements that can be used by the four dialects. Some of these elements are simple textual elements, such as *purposes*, *designation*, *description*, *history*, *surface* or the decimal element *dragCoefficient*. Other elements contain attributes, such as the ones shown in Fig. B.1(a) through B.1(f) (one attribute), and also in Fig. B.2(a) through B.2(c) (two attributes).

Some additional elements included in this file are depicted below. Figure B.3(a) shows the *ContactPerson* element, used by both SDL to specify the contact person of a base of operations and TDL to specify the contact person of a team. Figure B.3(b) shows the *Availability* element, used by the *Area* element and also in SDL for the base of operations. Figure B.4(a) shows the *Dimensions* element, used to specify vehicle payload dimensions in SDL and sensor dimensions in TDL. Figure B.4(b) shows the RelativeLocation element, used in SDL to specify the relative location of each payload in respect to the geometrical center of the vehicle. Figure B.5(a) shows the *Polygon* element and Fig. B.6(b) shows the *Circle* element, both used by the *Area* element, shown in Fig. B.5(b). Figure B.6(a) shows the *Coordinates* element, used by elements of SDL, DDL and MDL and by the *Location* element, shown in Fig. B.8(a). Figure B.7(a) shows the *TimeInterval* element, while Fig. B.7(b) shows the *TimePoint* element. Finally, Fig. B.8(b) and B.8(c) show the Mobility and Medium elements, respectively.

[1]The complete schema files are available from `http://www.fe.up.pt/~dcs/phd/`

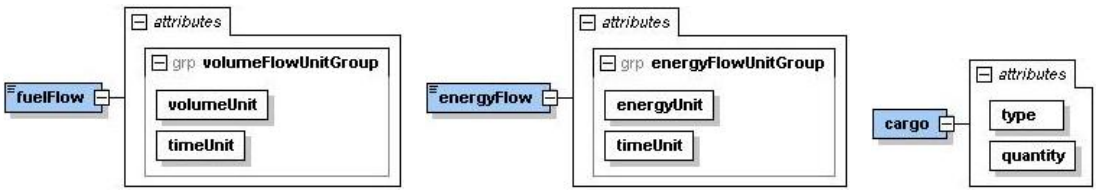


(a) Weight (b) Heading

(c) Quantity (d) Sense

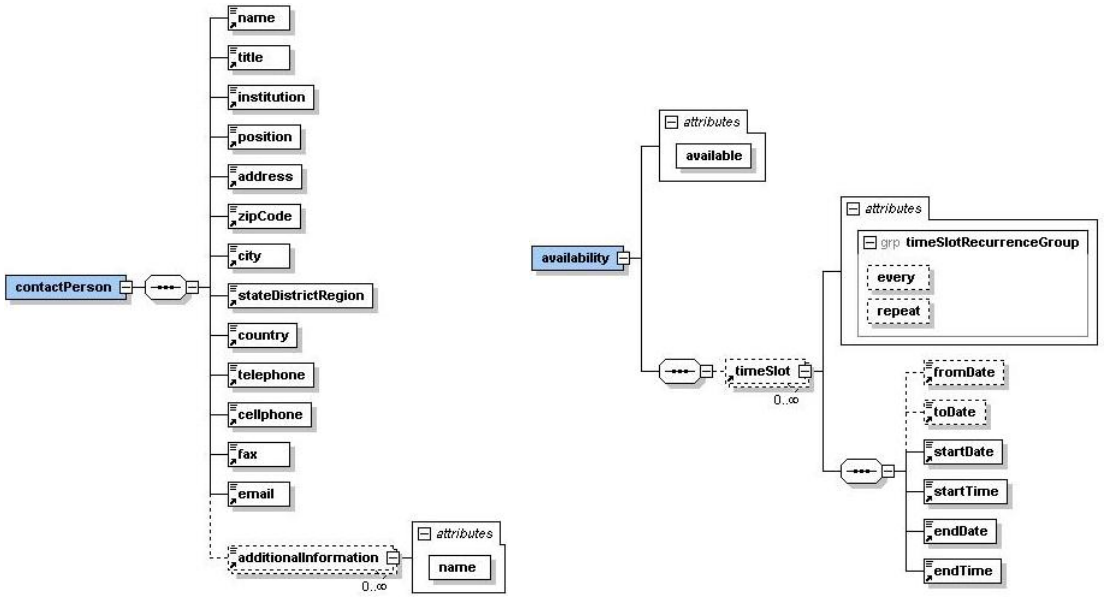
(e) MaxDepth (f) MaxVerticalVelocity

Figure B.1: Weight, Heading, Quantity, Sense, MaxDepth and MaxVerticalVelocity Elements



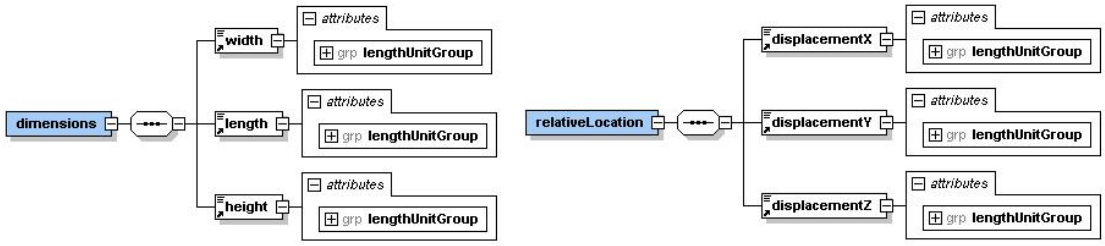
(a) FuelFlow (b) EnergyFlow (c) Cargo

Figure B.2: FuelFlow, EnergyFlow and Cargo Elements



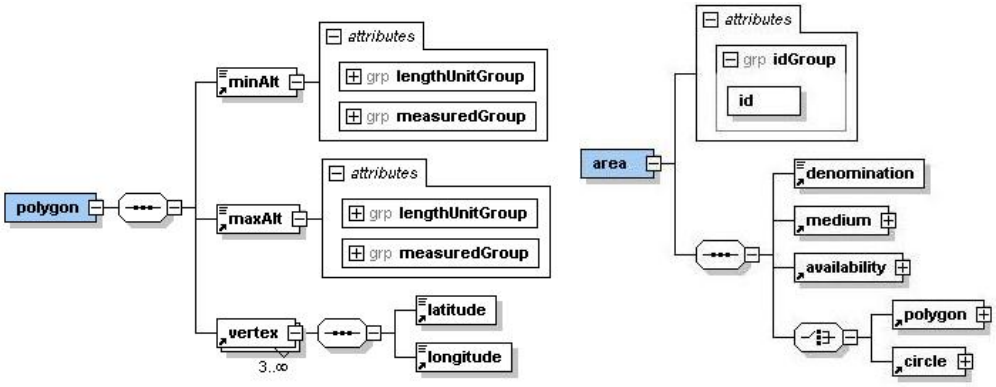
(a) ContactPerson (b) Availability

Figure B.3: ContactPerson and Availability Elements



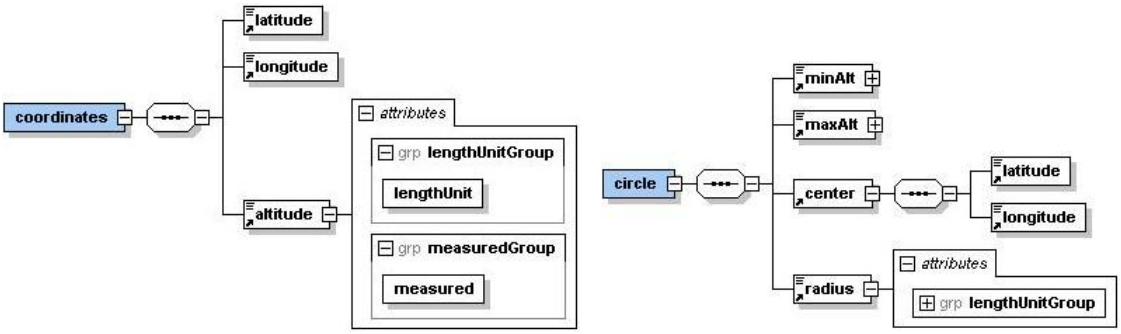
(a) Dimensions (b) RelativeLocation

Figure B.4: Dimensions and RelativeLocation Elements



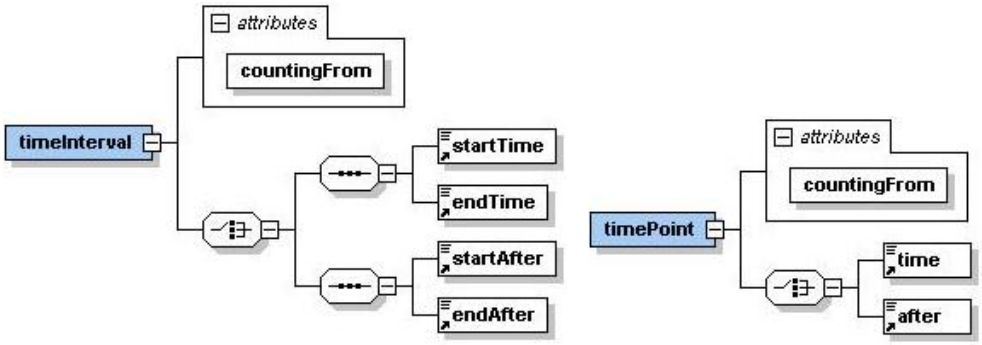
(a) Polygon (b) Area

Figure B.5: Polygon and Area Elements



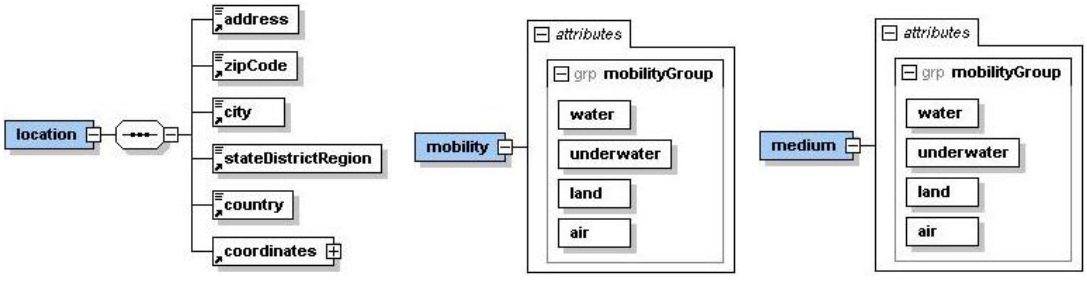
(a) Coordinates (b) Circle

Figure B.6: Coordinates and Circle Elements



(a) TimeInterval (b) TimePoint

Figure B.7: TimeInterval and TimePoint Elements



(a) Location (b) Mobility (c) Medium

Figure B.8: Location, Mobility and Medium Elements

B.2 Scenario Description Language

The SDL definition is shown in the following images. Figure B.9(a) shows the root element, *Scenario*. Figure B.9(b) shows the *BaseOfOperations* element, which is detailed in the following figures.

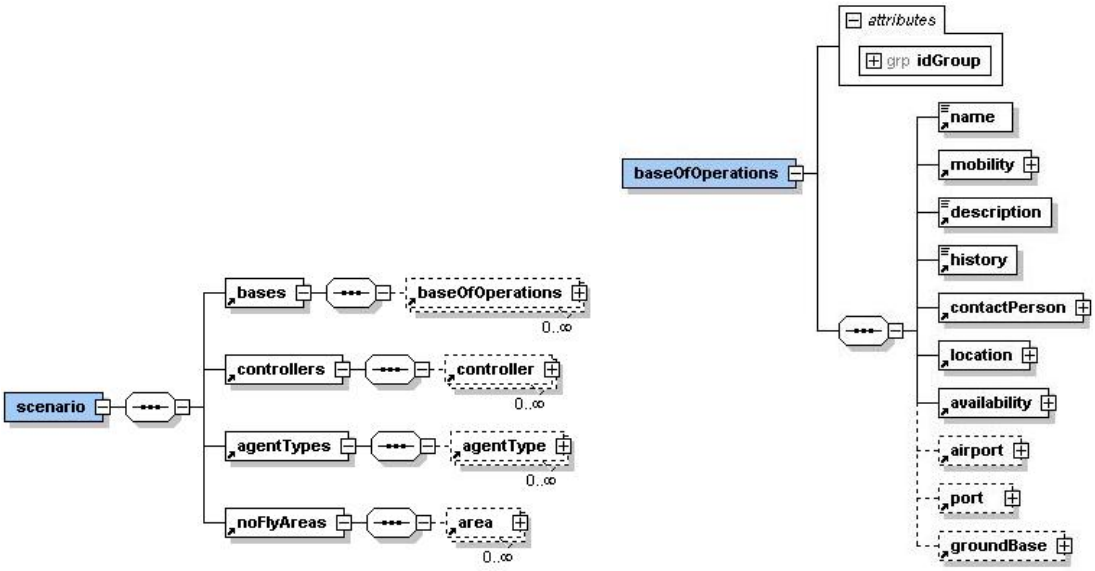
The *Airport* element is depicted in Fig. B.10, and Fig. B.11 to B.15 show the several elements that compose the airport – runway, taxiway, helipad, parking, hangar and utilities.

The *Port* element is shown in Fig. B.16(a), and Fig. B.16(b) to B.18(b) show the several elements that compose the port – waterway, quay, mooring, slipway and dry dock.

The *GroundBase* element is depicted in Fig. B.19(a), and Fig. B.19(b) to B.20(b) show the elements that compose the ground base – road, parking and garage.

Figure B.21 shows the *Controller* element, which allows for the specification of traffic control agents.

Finally, Fig. B.22 shows the *AgentType* element, and Fig. B.23 to B.25 show its constituent elements – RealAgentType, Physical and Performance.



(a) Scenario (b) BaseOfOperations

Figure B.9: Scenario and BaseOfOperations Elements

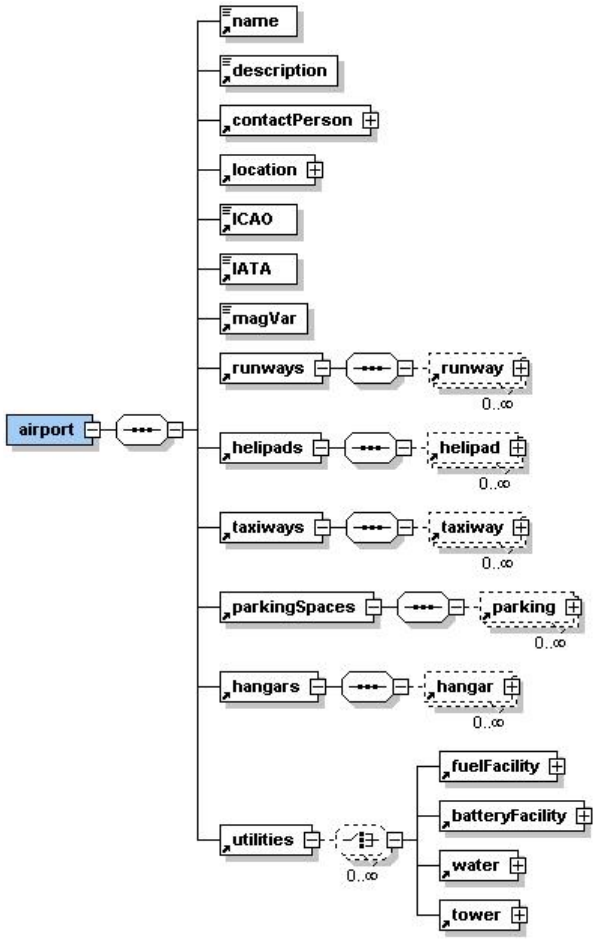


Figure B.10: Airport Element

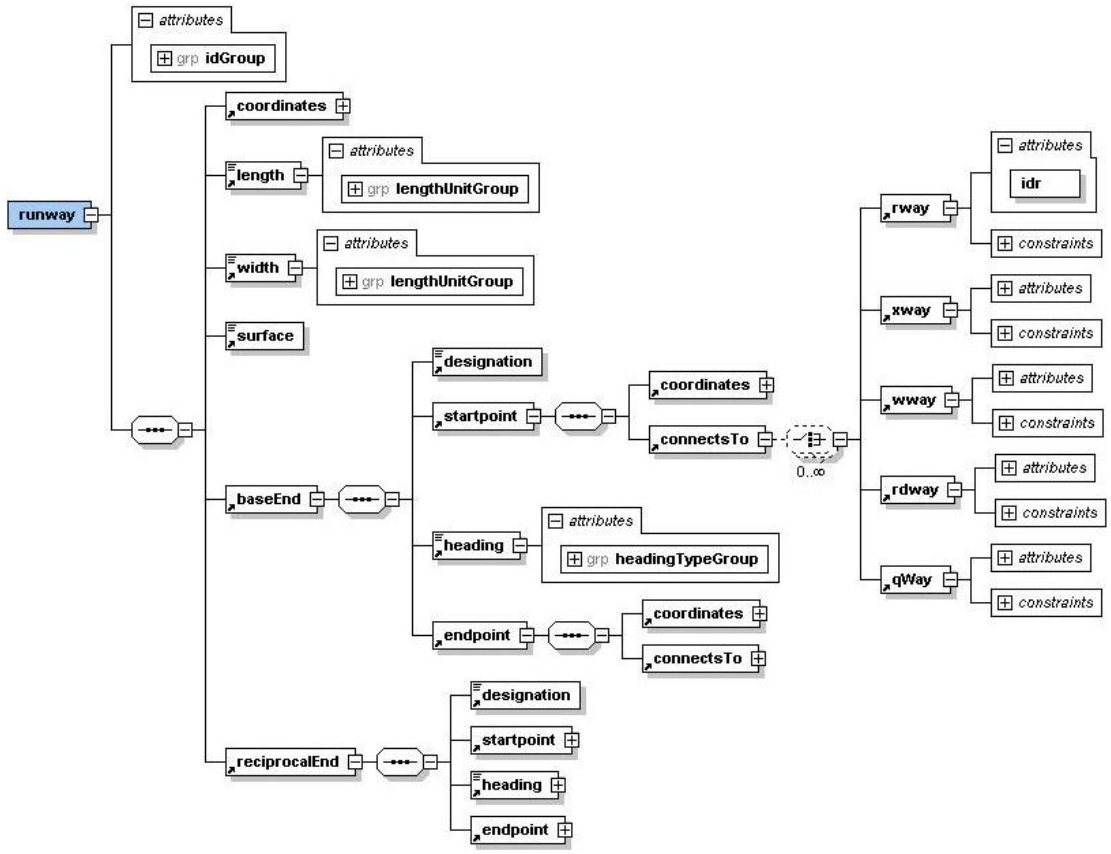
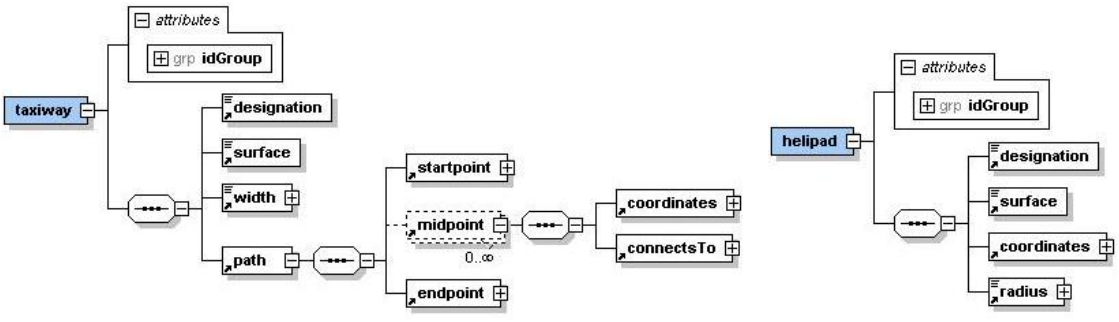


Figure B.11: Runway Element



(a) Taxiway (b) Helipad

Figure B.12: Taxiway and Helipad Elements

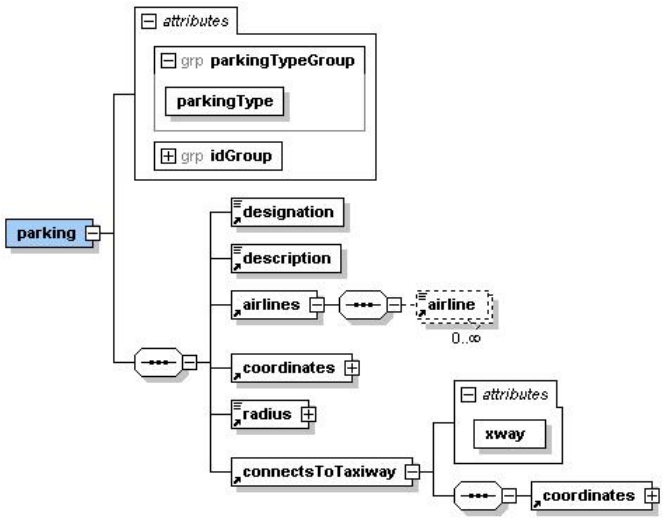


Figure B.13: Parking Element

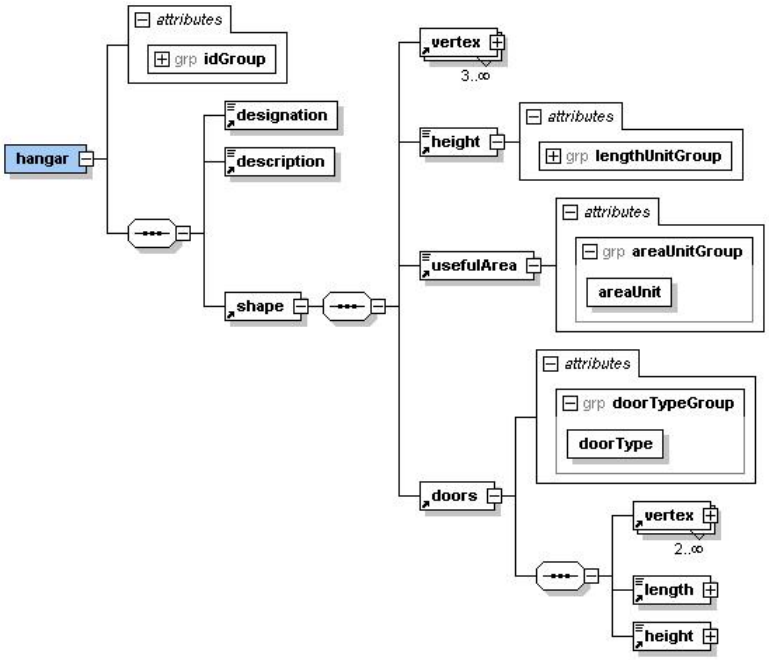


Figure B.14: Hangar Element

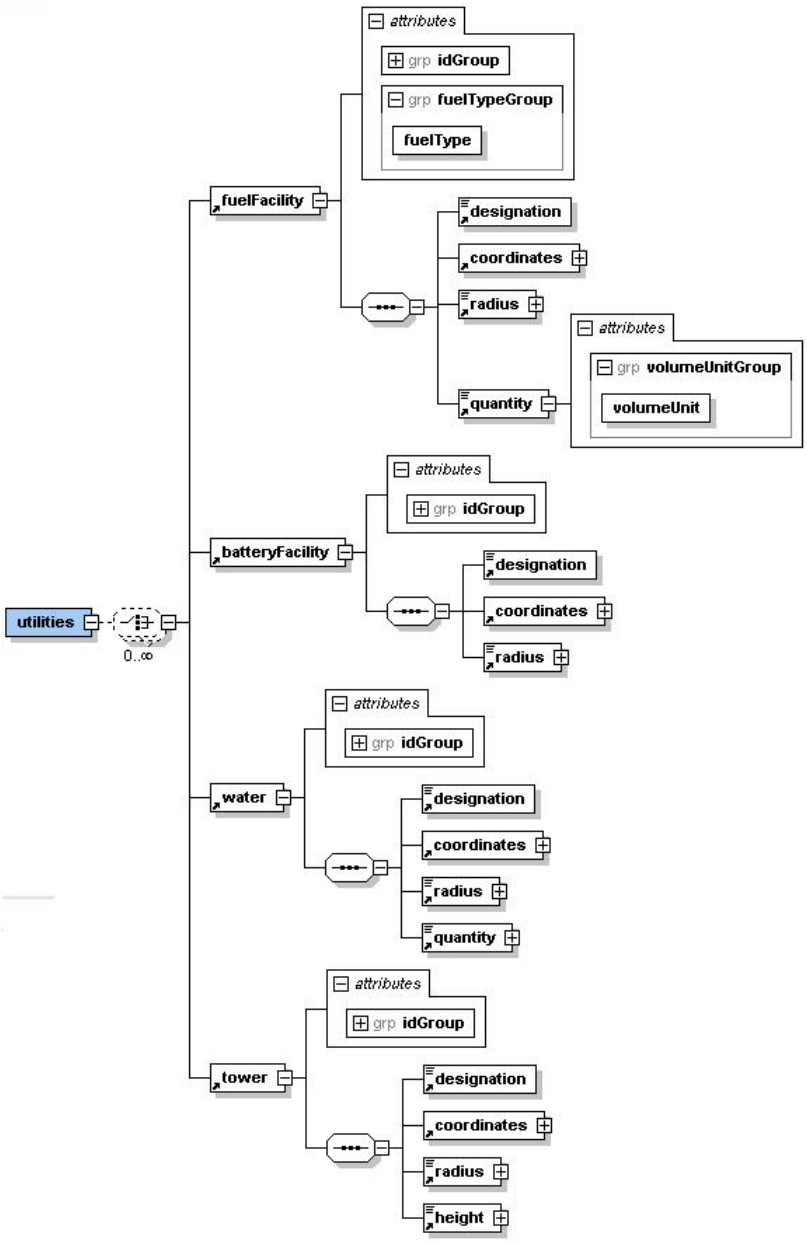
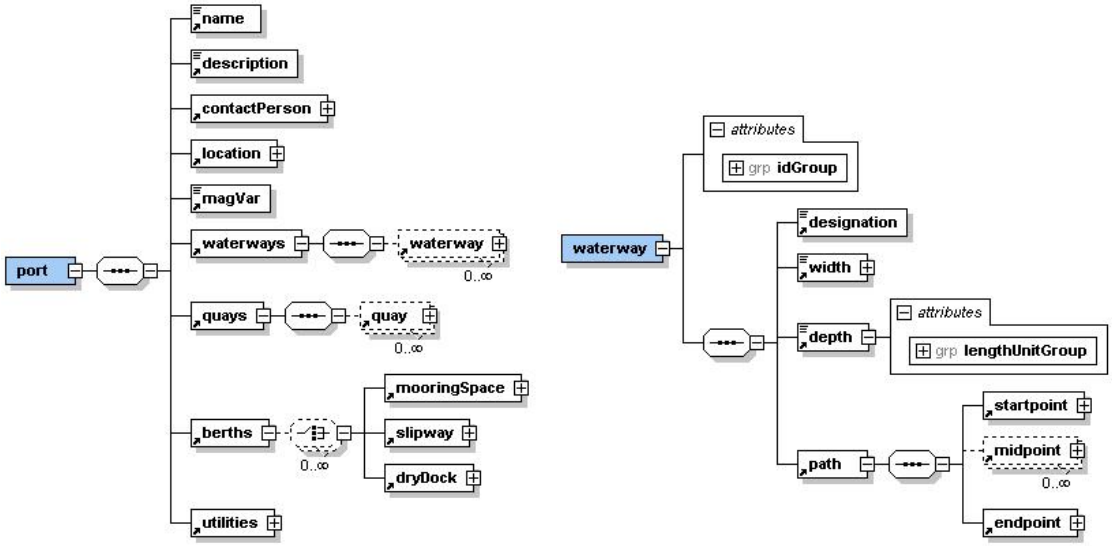
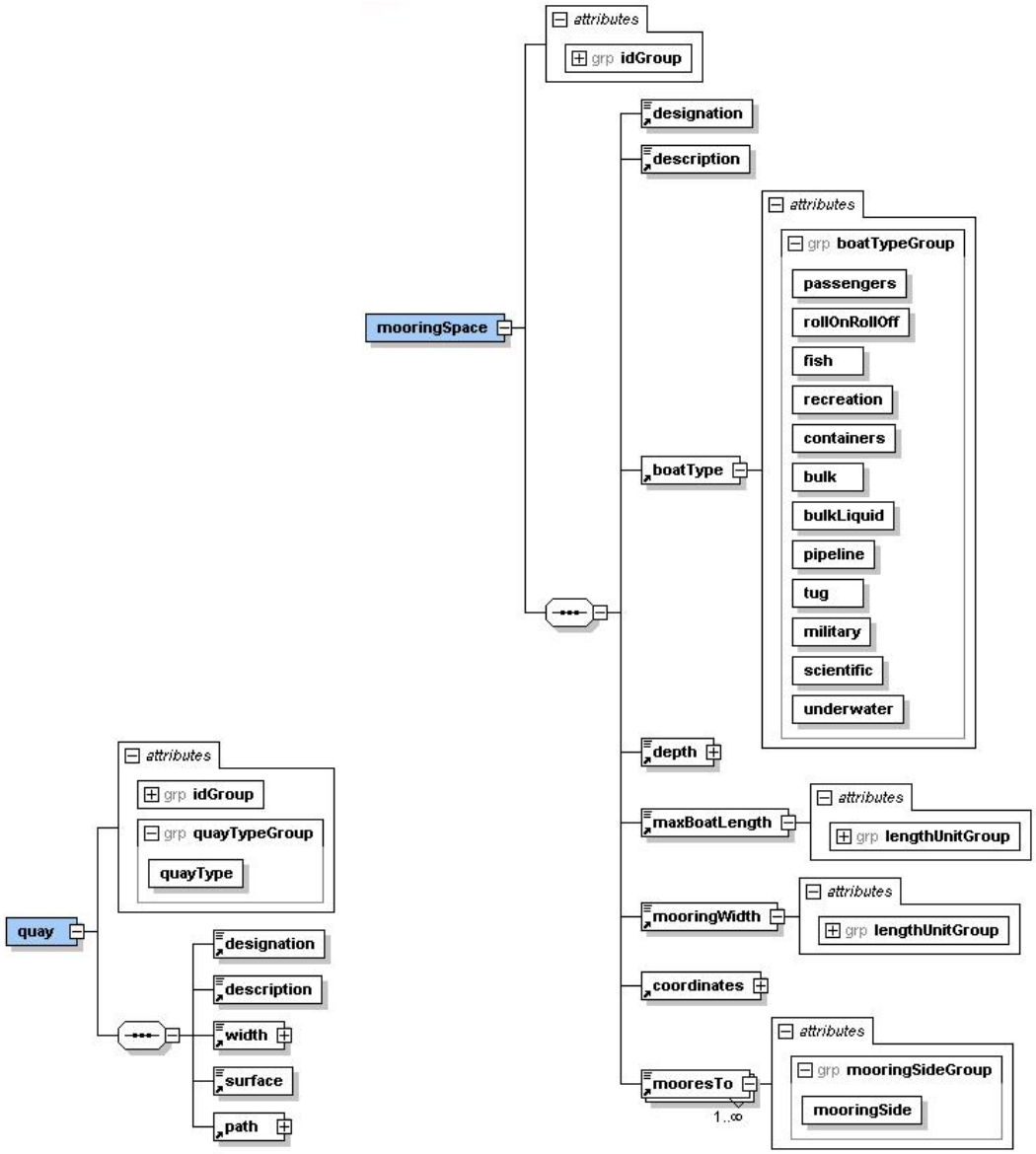


Figure B.15: Utilities Element



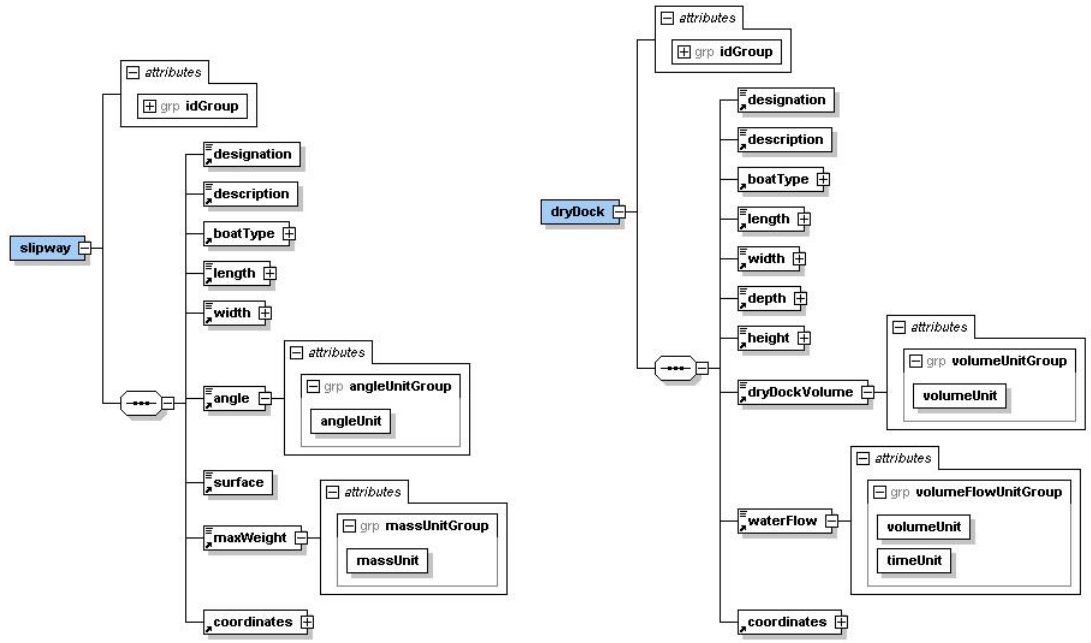
(a) Port (b) Waterway

Figure B.16: Port and Waterway Elements



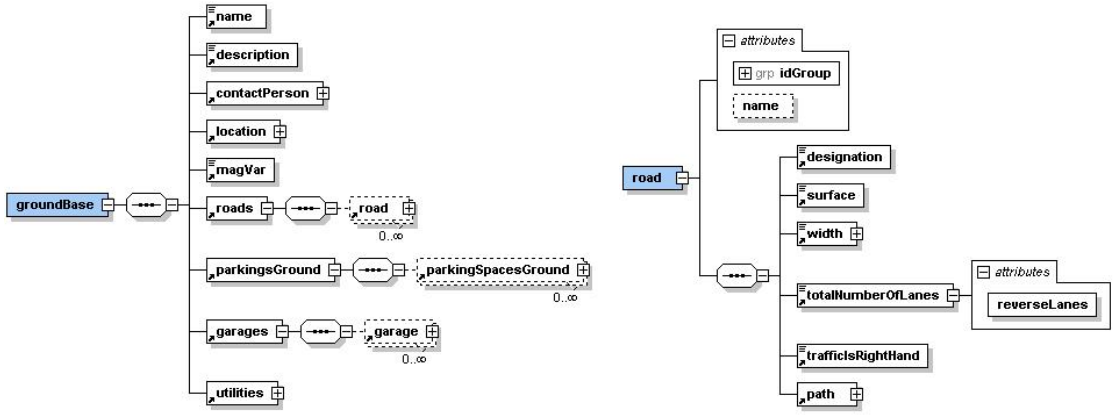
(a) Quay (b) Mooring

Figure B.17: Quay and Mooring Elements



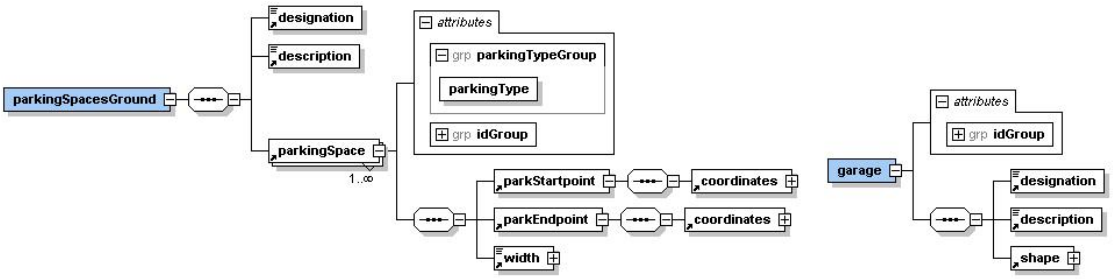
(a) Slipway (b) DryDock

Figure B.18: Slipway and DryDock Elements



(a) GroundBase (b) Road

Figure B.19: GroundBase and Road Elements



(a) ParkingGround (b) Garage

Figure B.20: ParkingGround and Garage Elements

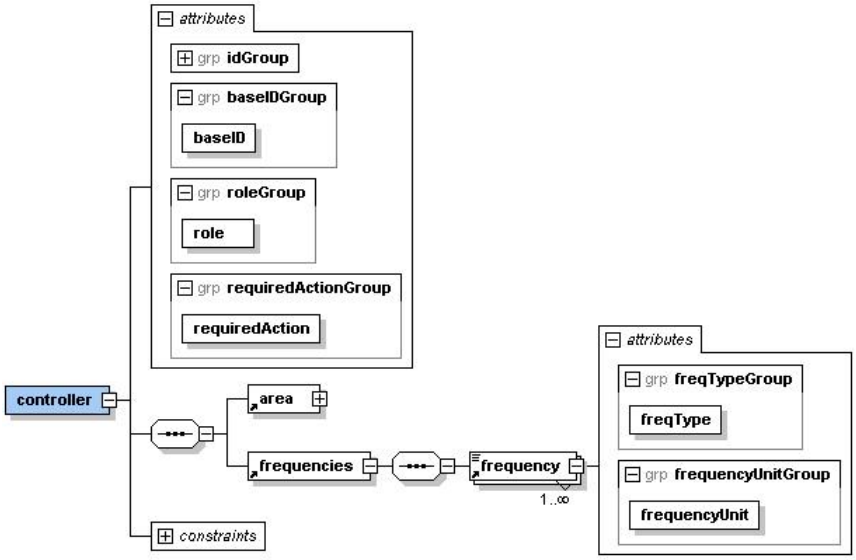


Figure B.21: Controller Element

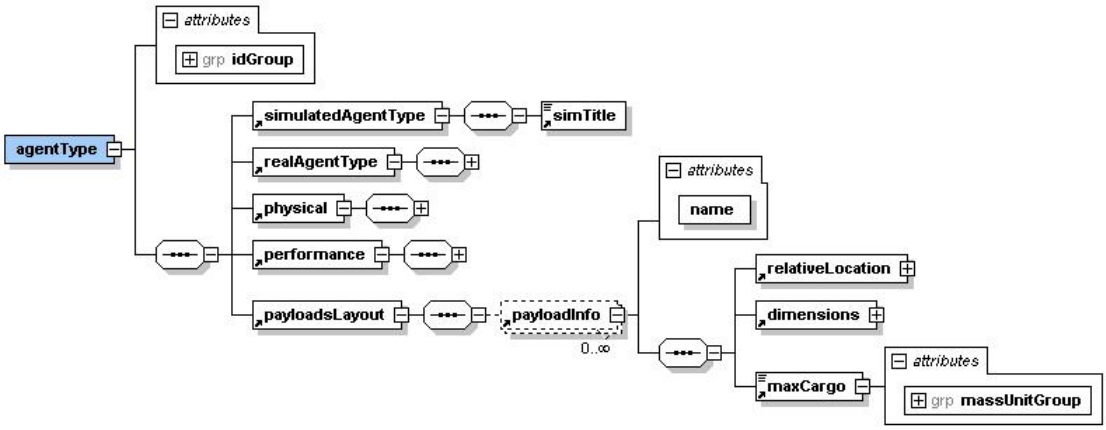


Figure B.22: AgentType Element

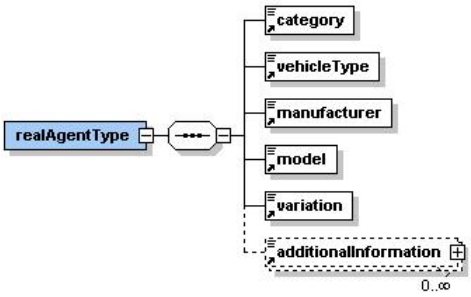


Figure B.23: RealAgentType Element

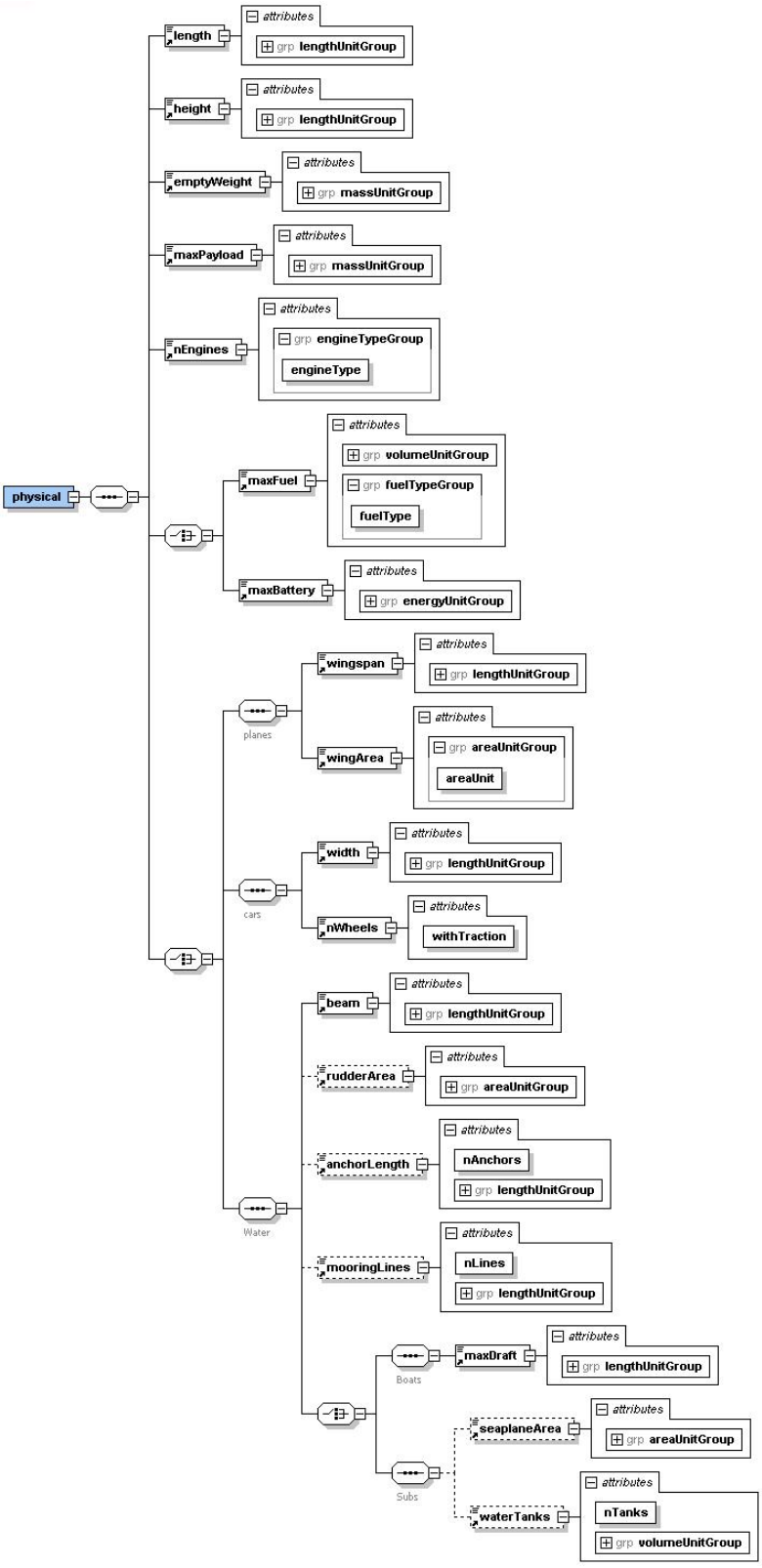


Figure B.24: Physical Element

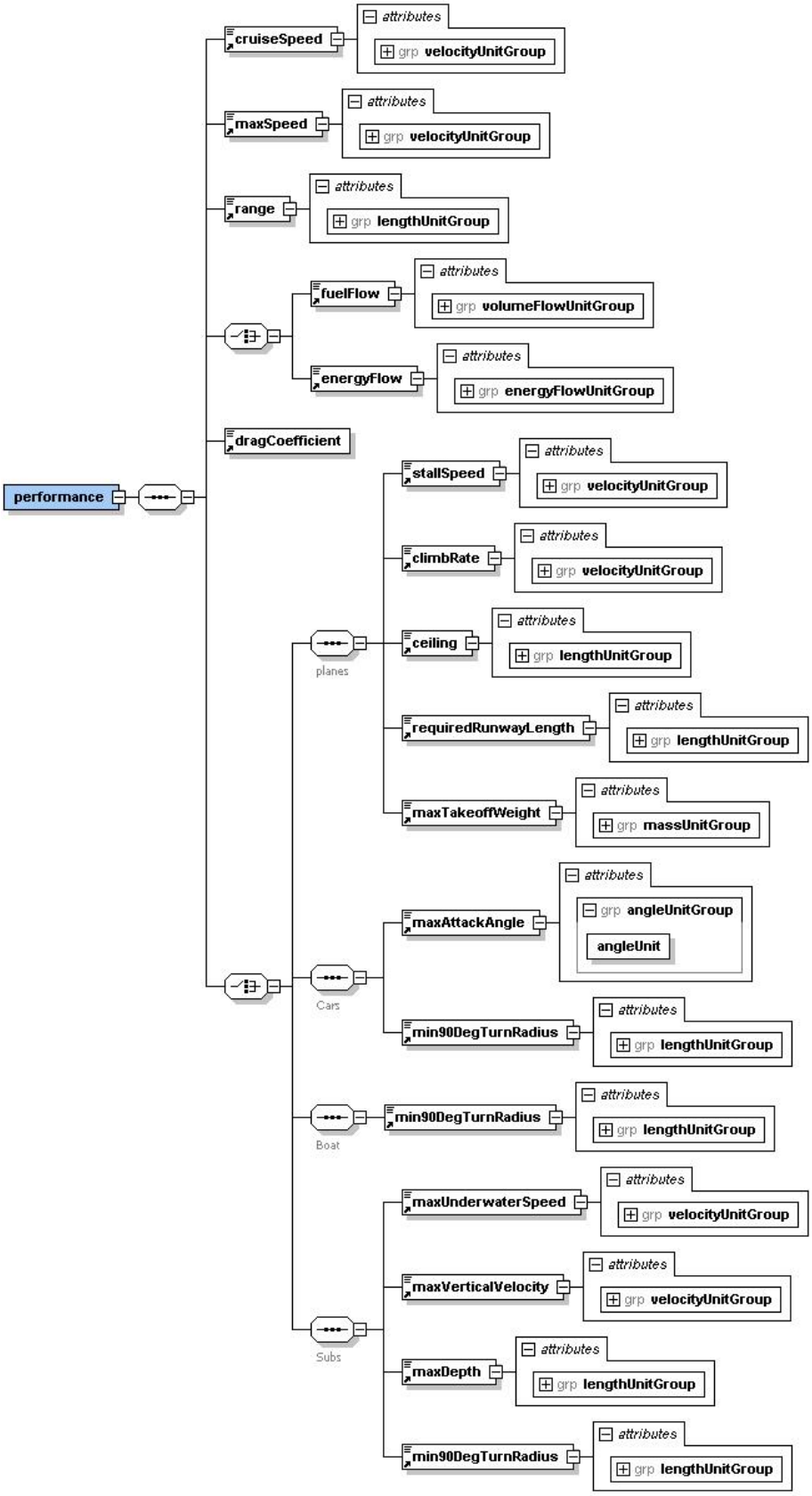


Figure B.25: Performance Element

B.3 Team Description Language

The TDL definition is shown in the following images. Figure B.26 shows the root element, *Teams*. Figure B.27 depicts the *Agent* element, which is detailed in the following figures – Fig. B.28, B.29 and B.30.

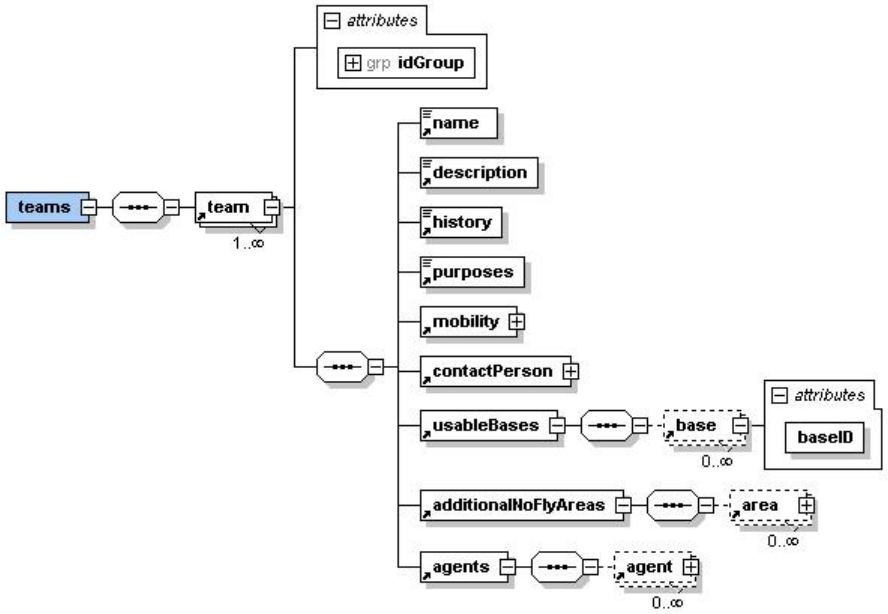


Figure B.26: Teams Element

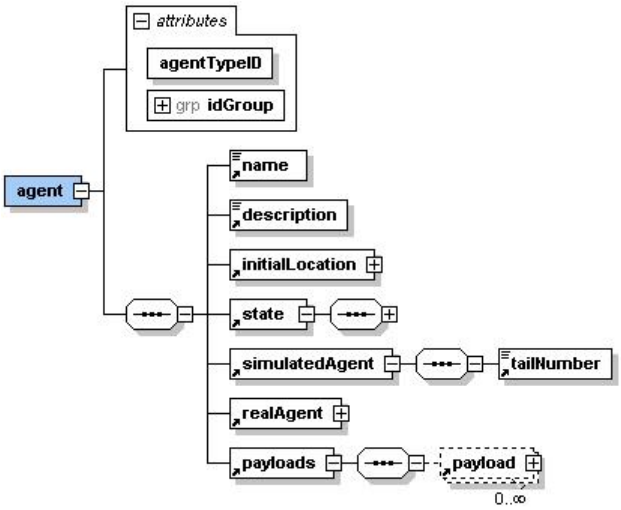


Figure B.27: Agent Element

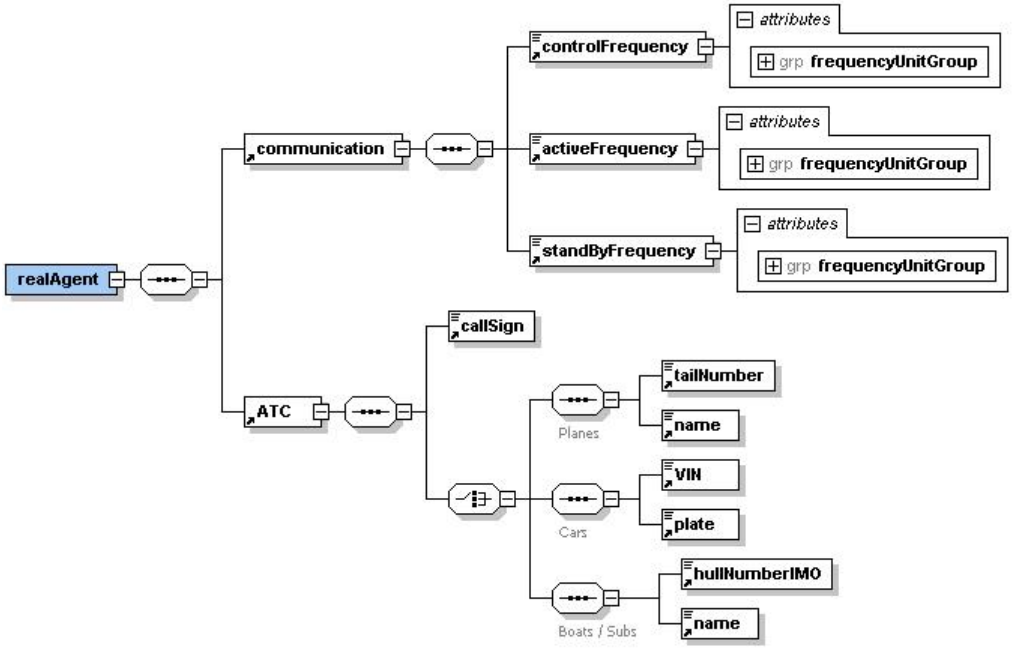


Figure B.28: RealAgent Element

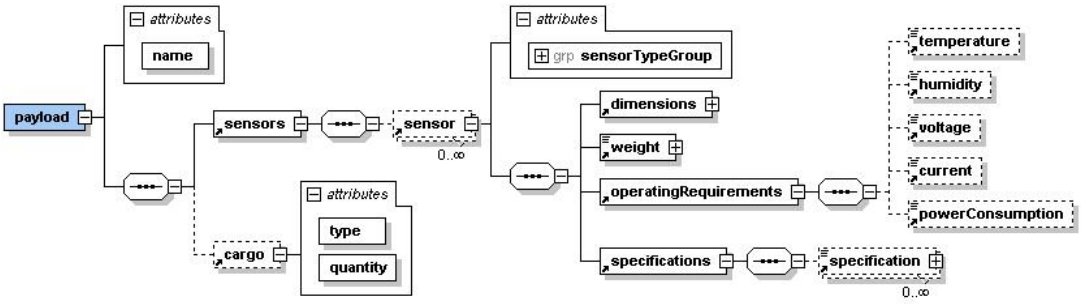


Figure B.29: Payload Element

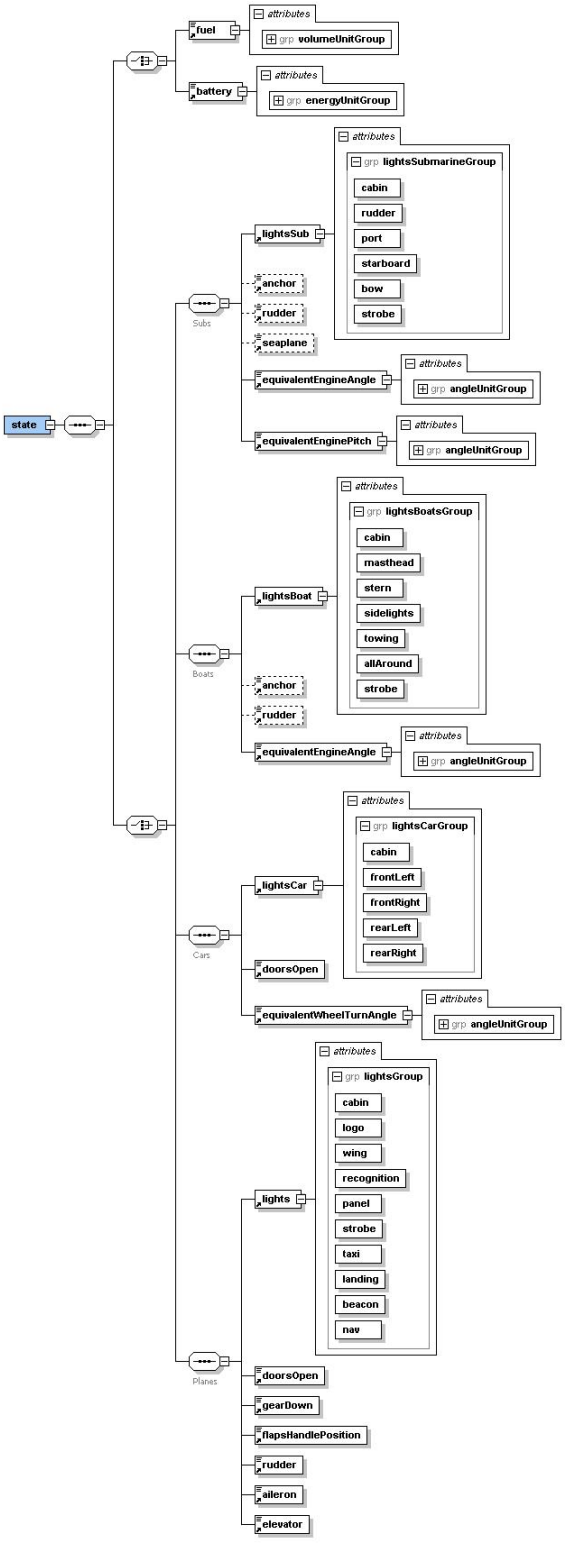


Figure B.30: State Element

B.4 Disturbance Description Language

The DDL definition is shown in the following images. Figure B.31 shows the root element, *Disturbances*, and Fig. B.32 to B.34 depict their composing elements.

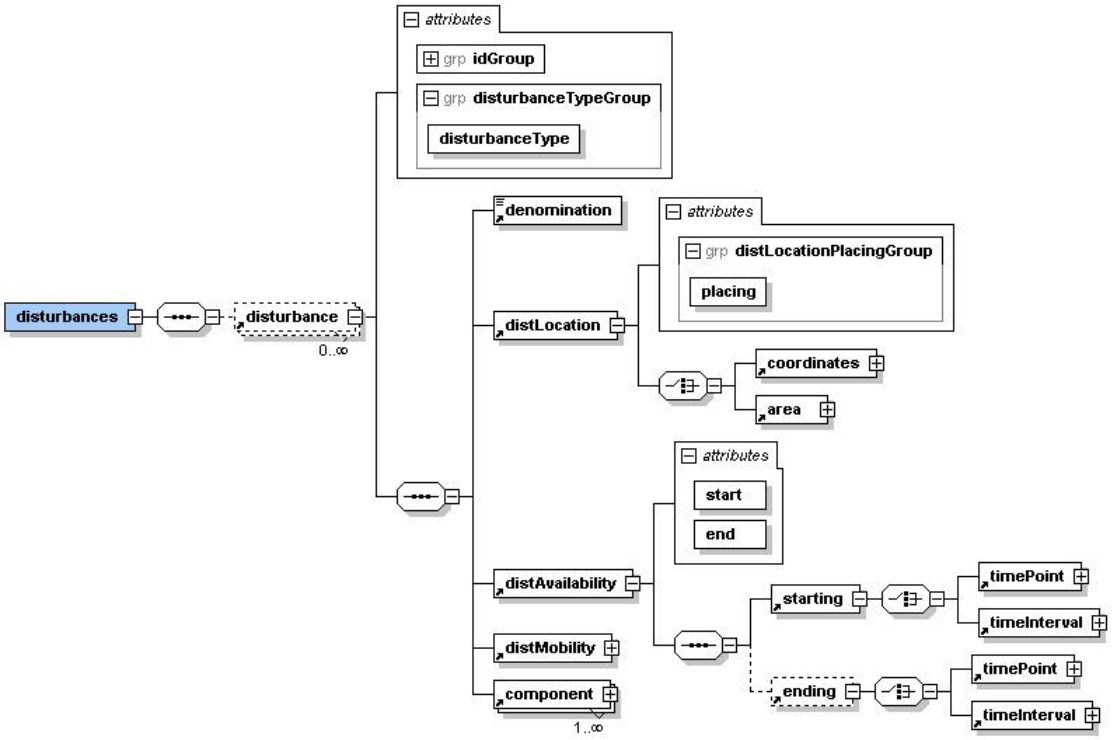


Figure B.31: Disturbances Element

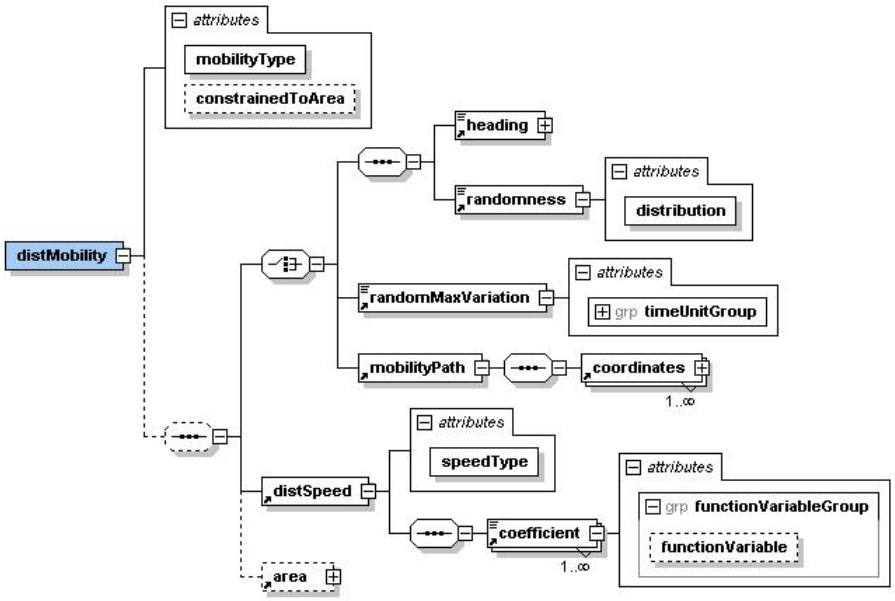


Figure B.32: Disturbance Mobility

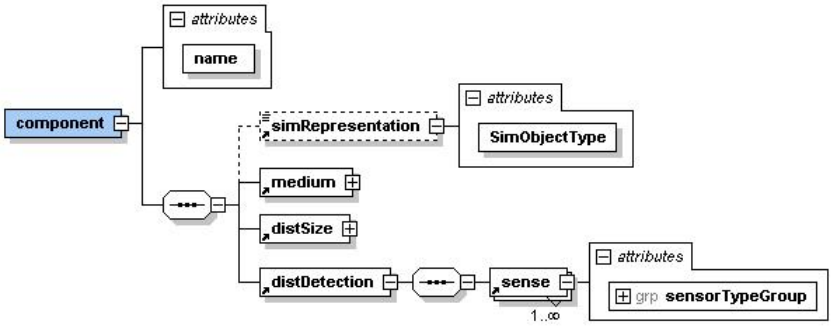


Figure B.33: Disturbance Component

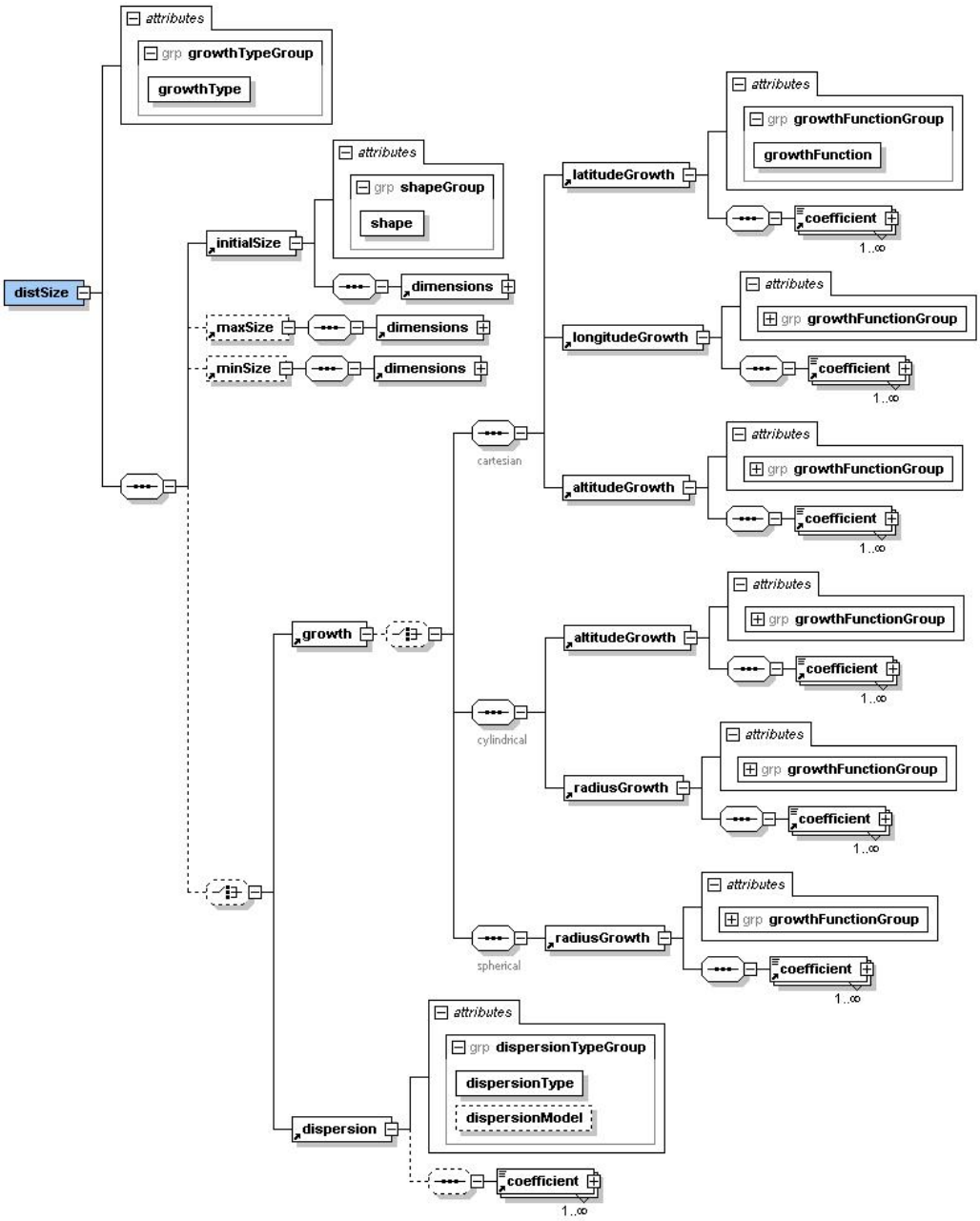


Figure B.34: Disturbance Size

B.5 Mission Description Language

The MDL definition is shown in the following images. Figure B.35 shows the root element, *Mission*, and Fig. B.36 to B.40 depict the composing elements of a mission phase.

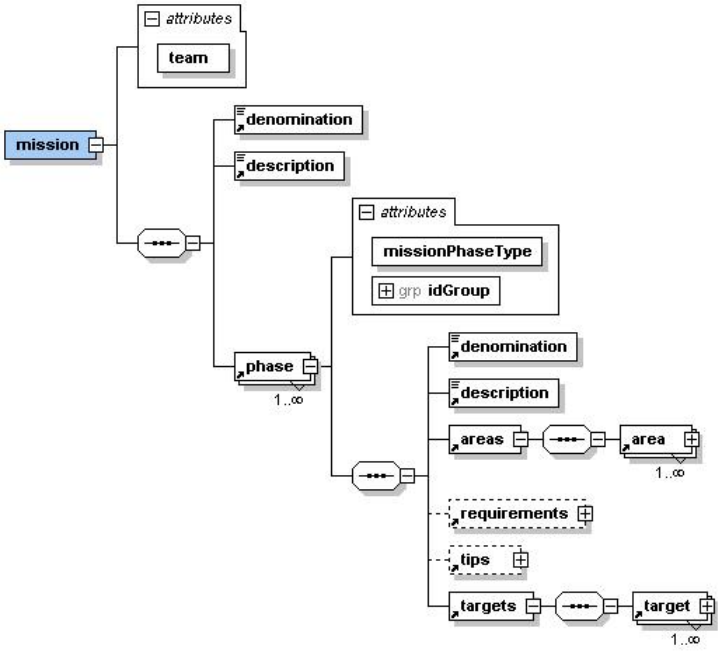


Figure B.35: Mission Element

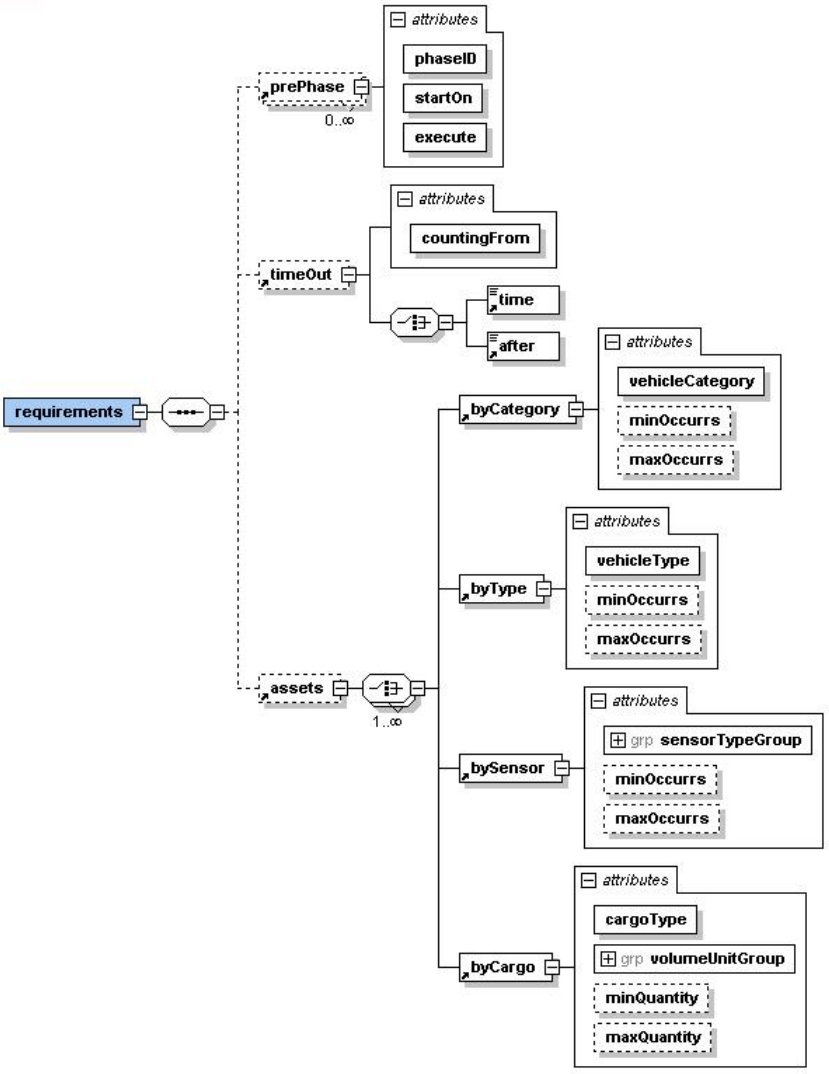


Figure B.36: Mission Requirements

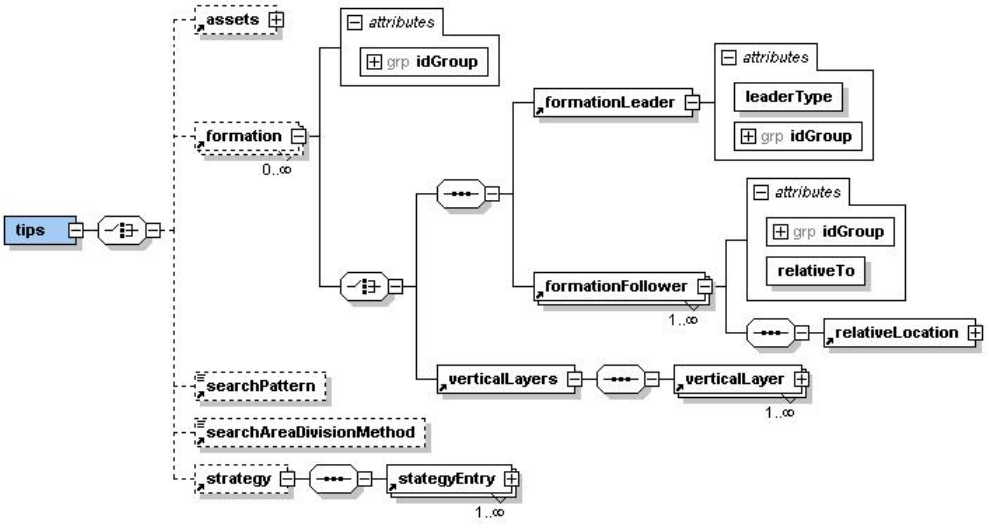


Figure B.37: Mission Tips

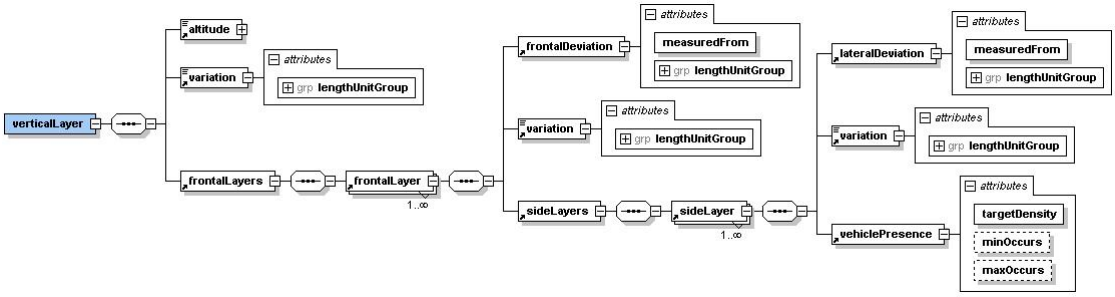


Figure B.38: Layers Element

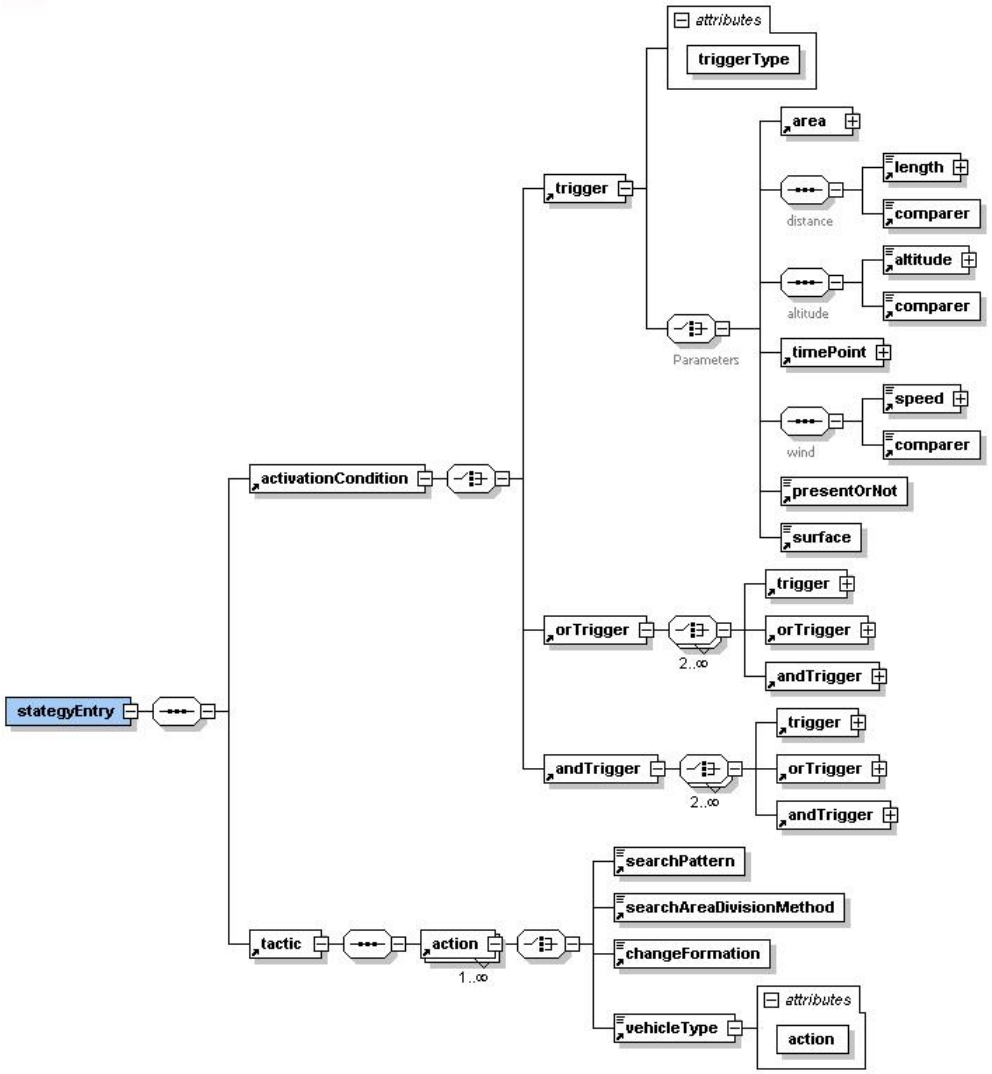


Figure B.39: Strategy Element

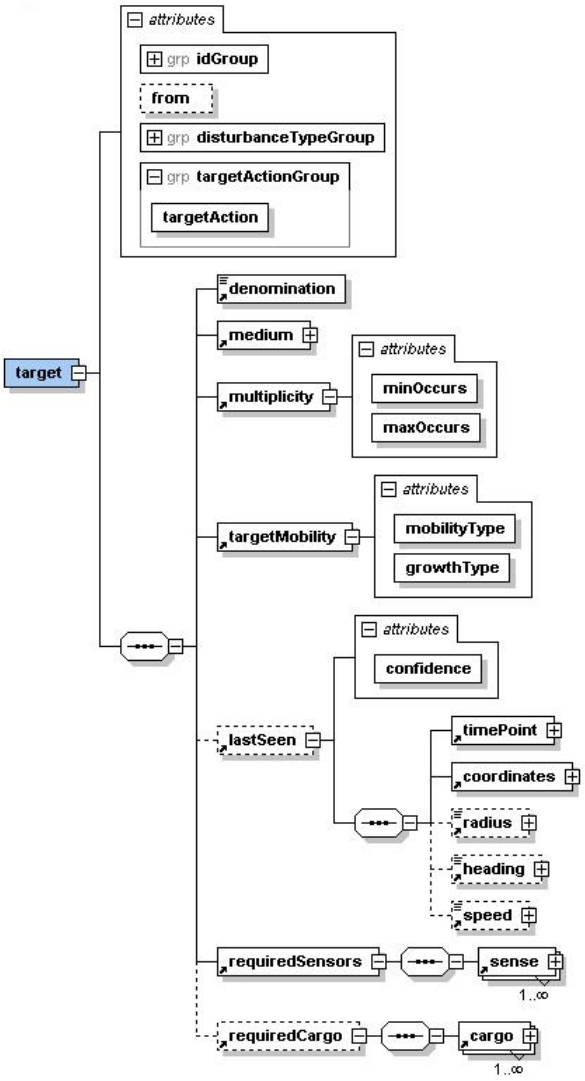


Figure B.40: Target Element