FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

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

Daniel Augusto Gama de Castro Silva

2011

U.PORTO

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 approach. Tests 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 realizam em locais que são perigosos ou inacessíveis para seres humanos. São também usados quando seria impraticável usar veículos operados por seres humanos, devido à tensão nos recursos humanos causada por missões de longo-termo, 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 serem usados num conjunto limitado de missões. A maioria das especificidades de cada plataforma é também normalmente definida de uma forma ad-hoc.

Esta dissertação tem como objectivo fornecer uma plataforma integrada para a execução de um vasto conjunto de missões por um grupo de veículos robóticos móveis heterogéneos. Para tal, foram desenvolvidos 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. Estes dialectos foram categorizados como estáticos vs. dinâmicos e ainda de acordo com a sua orientação para o cenário ou a equipa. A Linguagem de Descrição de Cenário (SDL – *Scenario Description Language*) descreve os elementos estáticos do cenário em que a missão terá lugar. A Linguagem de Descrição de Equipa (TDL – *Team Description Language*) descreve a composição da equipa e algumas restrições específicas da equipa. A Linguagem de Descrição de Distúrbios (DDL – *Disturbances Description Language*) descreve todos os elementos (dinâmicos) anómalos ao ambiente, e que irão constituir alvos para as missões. A Linguagem de Descrição de Missões (MDL – *Mission Description Language*) descreve a missão que deve ser executada pela equipa, constituída por um conjunto de fases.

Finalmente, foi desenvolvida a plataforma que implementa a arquitectura definida, constituída por vários componentes, incluindo um simulador realista; um painel de controlo para que o operador possa especificar a missão e elementos relacionados; um agente para controlo centralizado de tráfego numa área localizada; um agente para controlo de cada veículo, responsável, entre outras coisas, pelo planeamento e execução da missão; e alguns outros componentes.

Esta implementação de plataforma e linguagens permite uma configuração flexível de missões, equipas e cenários para as missões. Utilizando um cenário de teste que inclui um cenário contendo dois aeroportos, uma equipa constituída por diversos aviões, dois distúrbios simples e uma missão de reconhecimento simples, vários componentes foram testados a operar em conjunto, de forma

a validar a abordagem. Testes com outros tipos de veículos também provaram a possibilidade de usar esta plataforma com veículos que não aviões, demonstrando que esta abordagem é válida e que a plataforma pode ser usada com veículos de diferentes tipos. Adicionalmente, a modularidade e flexibilidade da plataforma permitem que esta seja usada como plataforma de testes em diversas áreas de investigação.

Résumé

Dans les dernières années, le nombre de véhicules autonomes développés pour effectuer différentes tâches a considérablement augmenté. Tant dans les scénarios militaires et civils, les véhicules avec un niveau croissant d'autonomie sont utilisés pour exécuter les missions qui ont lieu dans un endroit qui est soit dangereux ou inaccessibles pour l'homme. Ils sont également utilisés quand il serait impossible d'utiliser des véhicules exploités par les humains, que ce soit en raison de la pression sur les ressources humaines causées par les missions de longue durée ou de car il serait tout simplement trop coûteux d'utiliser de telles ressources. De nombreux projets ont été développés qui font usage de ces véhicules pour une variété de fins. La plupart de ces projets utilisent un nombre limité variété de véhicules et / ou sont conçus pour être utilisés pour une gamme étroite de missions. La plupart des spécificités de chaque plateforme sont aussi généralement définie de manière ad hoc.

Cette thèse vise à fournir une plateforme intégrée pour l'exécution d'une large gamme de missions conjointes par un groupe de véhicules robotiques hétérogènes. Pour cela, les deux une plateforme et un ensemble de langues qui permettent la configuration d'une mission ont été développés.

Dans une première phase, une architecture générale pour les systèmes qui comprennent les véhicules autonomes a été conçu, en prenant en considération plusieurs exigences communes. Ce modèle général a ensuite été instancié dans une architecture plus spécifique, où plusieurs éléments interagissent pour permettre la définition et l'exécution des missions de coopération.

Après la conception de l'architecture de la plateforme, et afin de configurer tous les aspects concernant la mission, quatre dialectes XML ont été créés - le scénario, les équipes, les perturbations et les missions. Ces dialectes ont été classés comme statique ou dynamique et aussi en fonction de leur orientation vers le scénario d'exploitation ou de l'équipe. Le langage de description de scénario (SDL - *Scenario Description Language*) décrit les éléments statiques du scénario dans lequel la mission aura lieu. Le langage de description de l'équipe (TDL - *Team Description Language*) décrit la composition de l'équipe et certains contraintes spécifique d'équipe. Le langage de description de perturbations (DDL - *Disturbances Description Language*) décrit tous les éléments (dynamique) anormal de l'environnement qui constituent des cibles pour le langage de description de la mission. Le langage de description de mission (MDL - *Mission Description Language*) décrit une mission pour l'équipe à effectuer, constituée par un ensemble de phases.

Enfin, une plateforme mise en œuvre de l'architecture définie a été développé, constitué de plusieurs composants, y compris un simulateur réaliste; un panneau de contrôle pour l'opérateur de préciser la mission et des éléments connexes; un agent de contrôle du trafic de manière centralisée sur une zone localisée, une l'agent de contrôle de chaque véhicule, responsable, entre autres choses, pour la planification de mission et d'exécution; et certains autres composants.

Cette mise en œuvre de la plateforme et des langues permet une configuration souple des missions, équipe et scénario de mission. En utilisant un paramètre de test comprenant un scénario configuré avec deux aéroports, une équipe composée de plusieurs avions, deux perturbations simple et une simple reconnaissance mission, plusieurs composants ont été testés lorsqu'ils travaillent

ensemble, afin de valider les approche. Les tests avec d'autres types de véhicules a également démontré la faisabilité de l'utilisation de cette plateforme avec véhicules autres que des avions, montrant que l'approche est valide et la plateforme peut être utilisée avec différents types de véhicules. En outre, la modularité de la plateforme et la flexibilité lui permet d'être utilisé comme un banc d'essai dans plusieurs nouvelles directions de recherche.

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 vehicles of different 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 platform, such as Search and Rescue activities, several military operations, civilian surveillance (fire, pollution), as well as some commercial uses. 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 ones presented herein are:

1. Initial Contributions

    - **Generic Model** Associated with the developed platform, the model developed for generic multi-robot 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 and 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]. These languages are used to provide the platform with the necessary information for a mission to be performed by the several vehicles.

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.

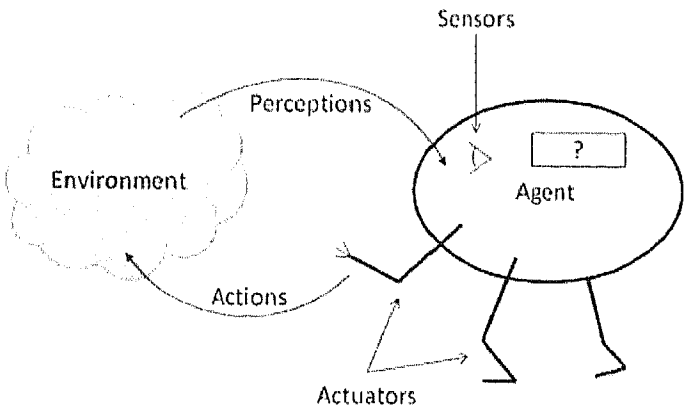


Figure 2.1: Generic Agent Schema (Adapted from [Russel & Norvig, 2002])

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 Multi-Agent 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 solved 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.

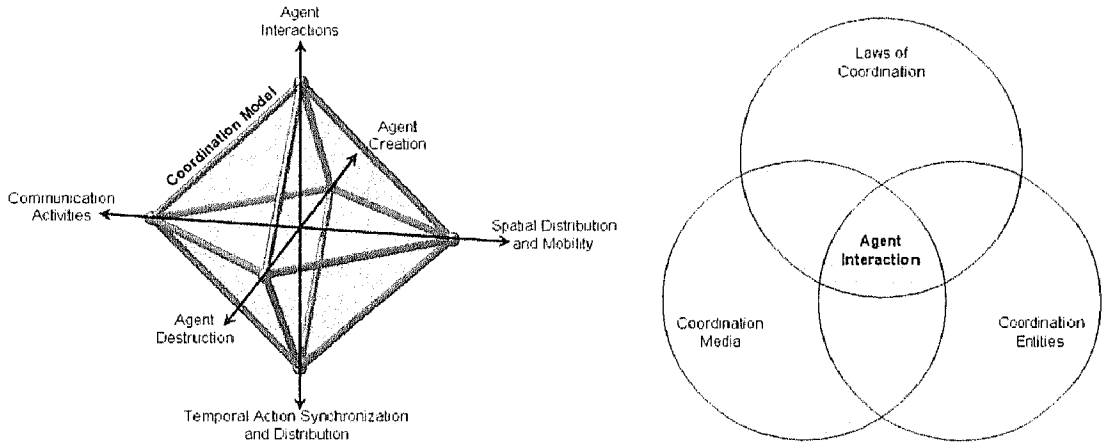


Figure 2.2: Coordination Model (Adapted from [Reis, 2003])

Early in the last decade, a new model was proposed by Silva and Demazeau, named Vowels [da Silva & Demazeau, 2002]. It considers four coordination levels – Agents, Environments, Interactions and Organizations – as well as the interactions between these four levels.

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 no developments to the platform were registered since then.

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.

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].

[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

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 Zeus, however, was released early in 2006, with no new announced developments since that time.

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.

[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/`

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 [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

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

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].

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 a 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].

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,

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

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.

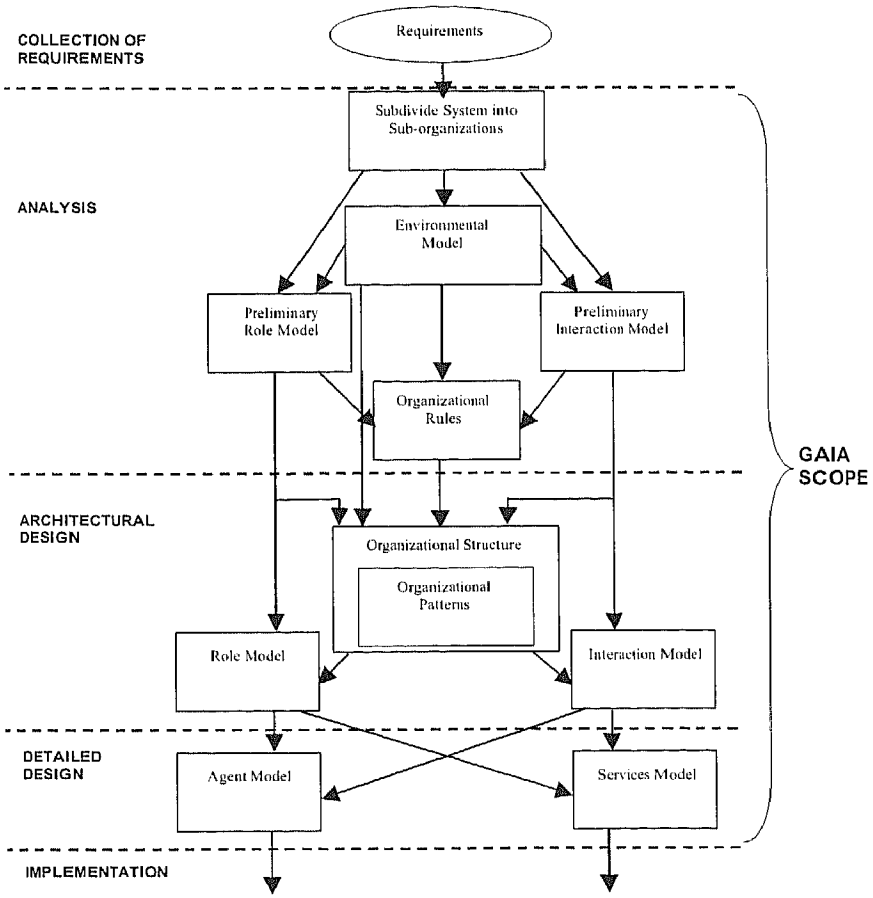


Figure 2.3: Models in the Gaia v.2 Methodology ([Zambonelli et al., 2003])

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

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

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).

- 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.

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

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 imitates 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 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

[14] More information available online at `http://www.sics.se/tac/page.php?id=13`

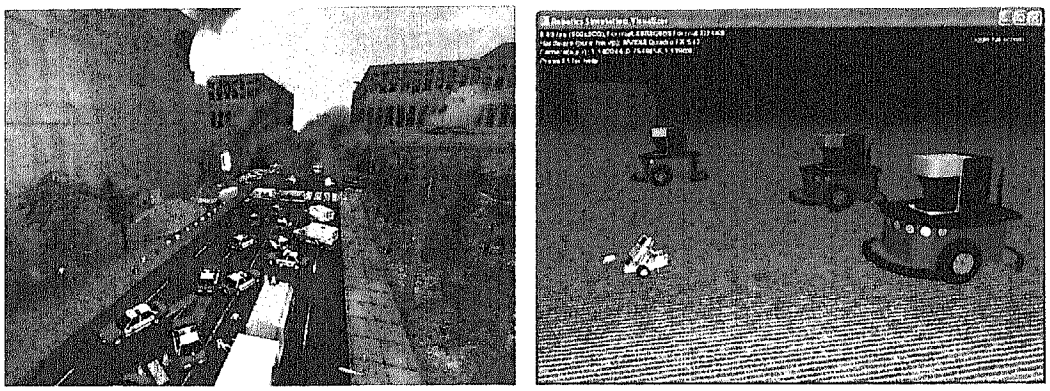
[15] More information about the Unreal Engine available from `http://www.unrealtechnology.com/`

[16] More information about the RoboCup Rescue available at `http://www.robocuprescue.org/`

[17] More information available from `http://www.microsoft.com/robotics/`

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 named 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].



(a) USARSim (b) Microsoft Robotics Studio

Figure 2.4: USARSim and Microsoft Robotics Studio

2.2.2 Fligh Simulation

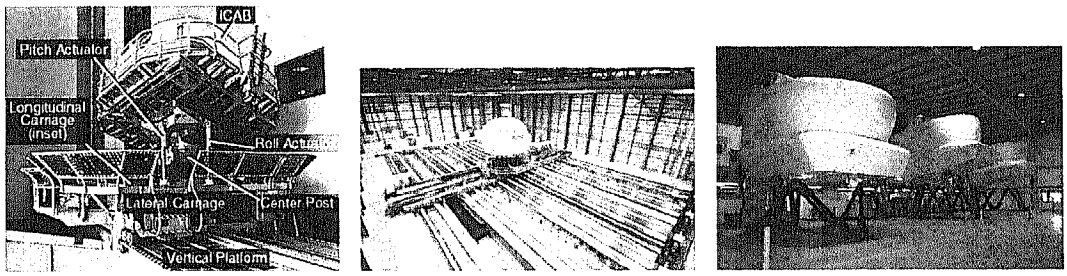
With one century of history [Page, 2000], flight simulators appear in a wide range of configurations, diverging in complexity, physical and/or simulated nature or orientation to professional training or entertainment (games). Early flight simulators consisted of human-operated constructions that simulated 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

[18]More information available from `http://playerstage.sourceforge.net/`

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] [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.

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

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

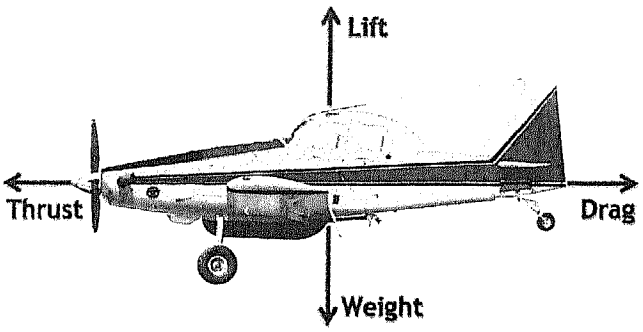


Figure 2.6: Simplified Forces of Flight

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):

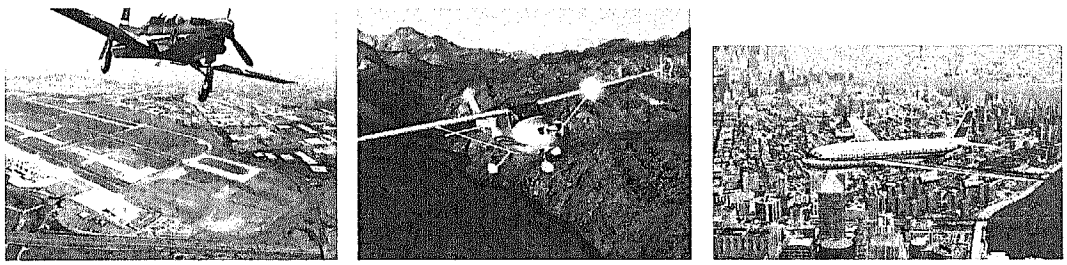
- **FlightGear.** FlightGear is a free, open-source, multi-platform, cooperative flight simulator, and the source code for the entire project is available and licensed under the GNU General Public License[23]. The goal of the FlightGear project is to create a sophisticated flight simulator framework 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.

[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

- **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 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 auto-pilot) 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].

Auto-pilot 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), auto-pilots 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].

Auto-pilot 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

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

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 $12km$ 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 $96km$ 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 auto-pilot 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 auto-pilot systems appeared only about a decade later; and b) boats and cars existed long before aircraft and auto-pilot systems for these vehicles were only later developed, it is easy to understand why aircraft auto-pilots are more advanced than auto-pilot systems for other vehicle types. In fact, several commercial aircraft feature auto-pilot 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 auto-pilot 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, auto-pilot systems tend to also decrease in size, which means that additional challenges need to be faced.

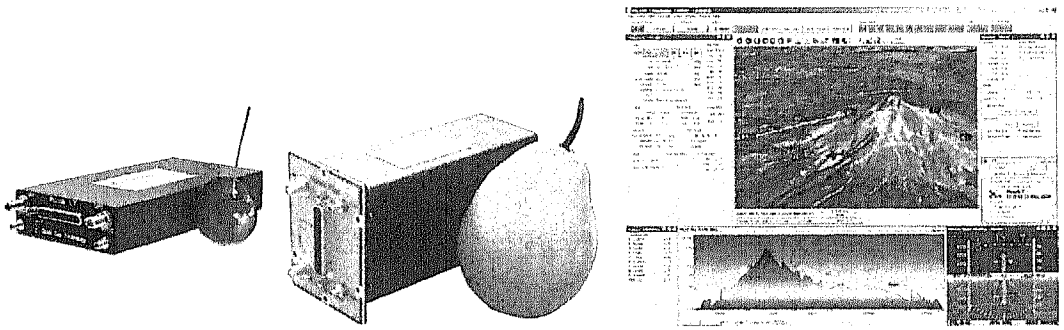
There is a variety of auto-pilots commercially available for small unmanned aircraft, usually comprised of light-weight hardware to connect to the aircraft, and a control station, also often including some software to interact with the system. These auto-pilots range from simple one-axis controllers 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 auto-pilot systems with the aircraft has to be carefully executed, as to calibrate the system to the aircraft in question [Erdos & Watkins, 2008].

Auto-pilot 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 auto-pilot system is provided by MicroPilot[26], as miniature UAV auto-pilots, weighing as little as 28 grams. Another example is Piccolo, aimed at small aircraft,

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

and commercialized as a complete, off the shelf avionics system solution, including the core auto-pilot, 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.



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

Figure 2.8: Piccolo Hardware The Piccolo Command Center

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

[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.

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 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].

[28] More information available online from `http://www.blueangels.navy.mil/`
[29] More information available online at `http://www.asasdeportugal.com.pt/`

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].

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

[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/`

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 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

[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

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].

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.

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

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 an adapted version of 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 extending the Gaia methodology [Castro & Oliveira, 2008], an adapted version of 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 presented 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.

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

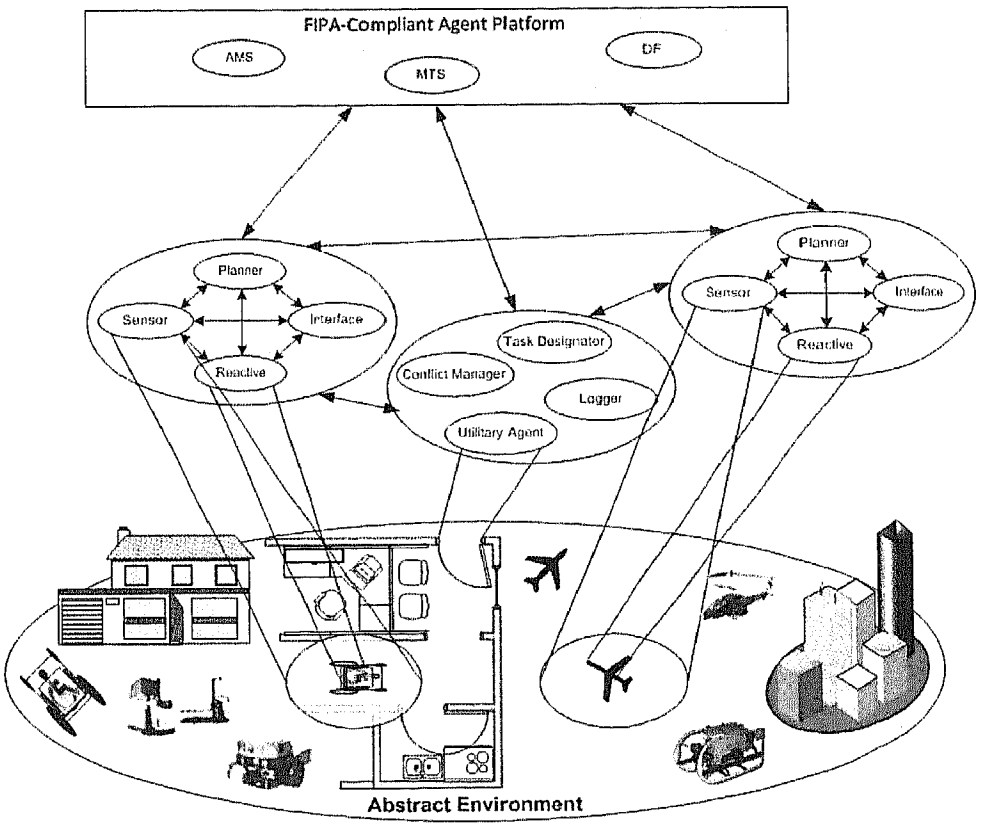


Figure 3.1: Generic Informal Platform Architecture

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 (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

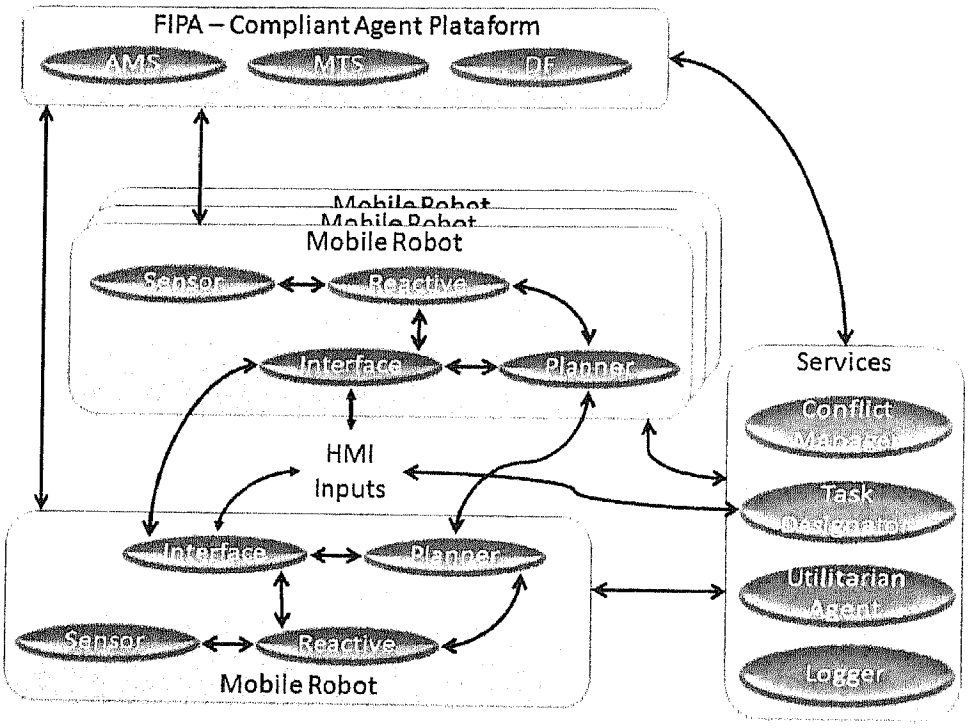


Figure 3.2: System Sub-Organizations as Groups

system implementing this generic model should better describe the environment it will operate on, and the possible particularities of such environment.

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.

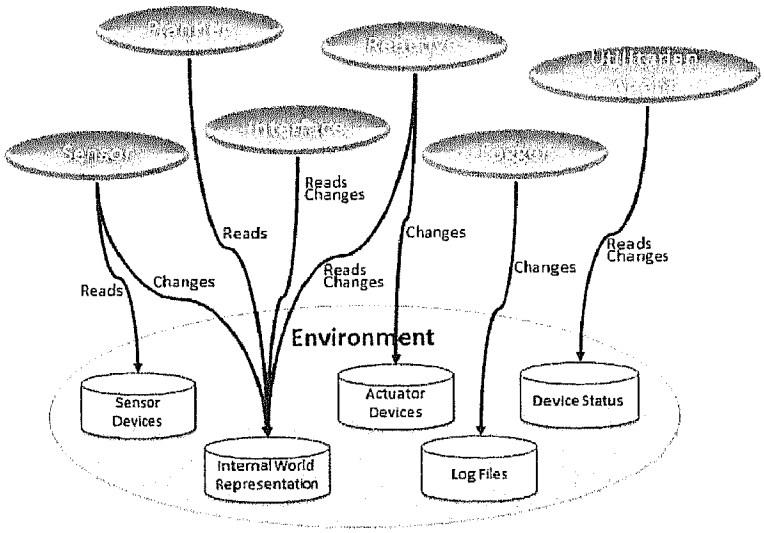


Figure 3.3: Environment Resources Diagram

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.

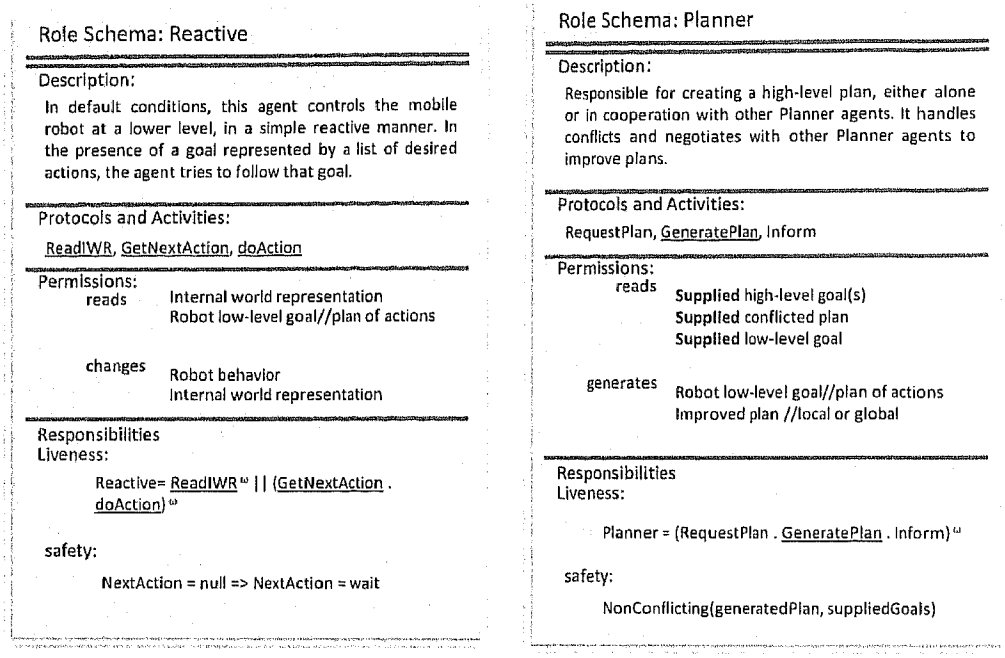
The Reactive role (see Fig. 3.4(a)) is also a basic role, and is responsible for all low level control, using the internal representation of the environment together with low level goals for determining action control.

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



(a) Reactive Role (b) Planner Role

Figure 3.4: Reactive and Planner Roles

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, 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.

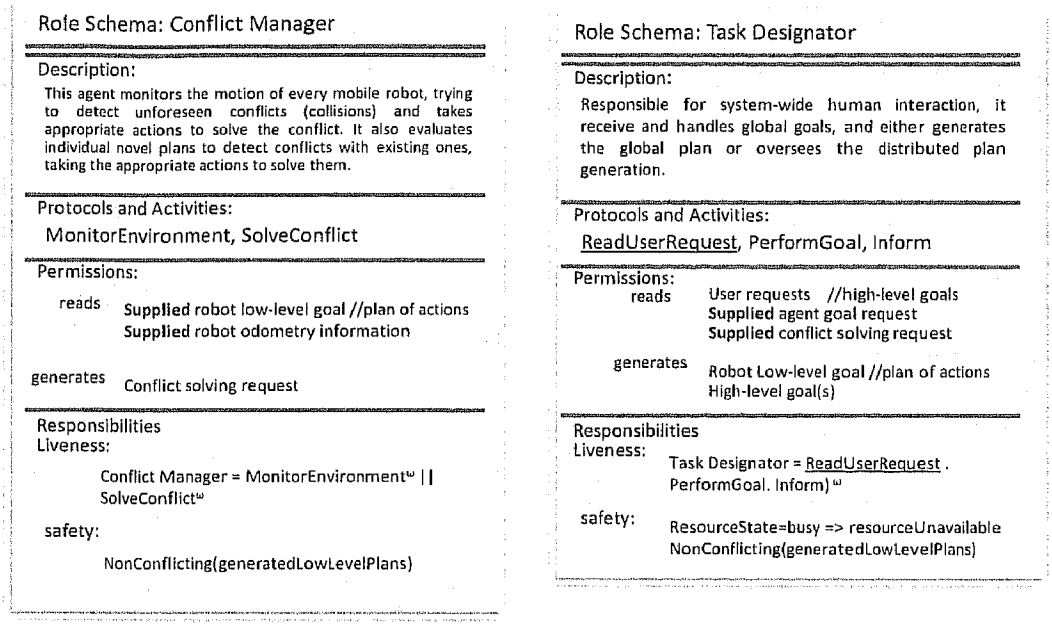
The Task Designator role (see Fig. 3.5(b)) is responsible for providing human actors with a means to interact with the system as a whole. This allows them to specify the missions that should be carried out by the system (either by a single robotic platform, or by multiple platforms).

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.



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

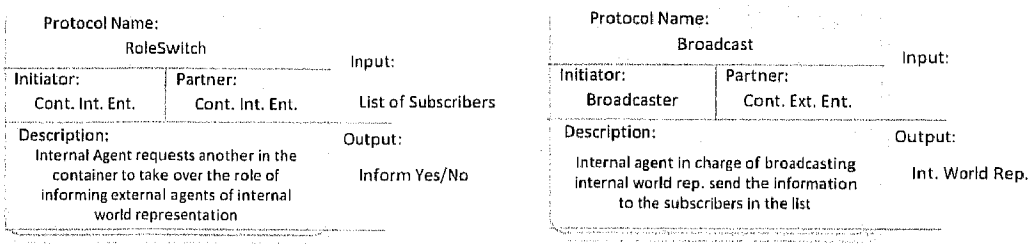
Figure 3.5: Conflict Manager and Task Designator Roles

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).

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



(a) Role Switch Protocol (b) Broadcast Protocol

Figure 3.6: Role Switch and Broadcast Protocols

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.

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.

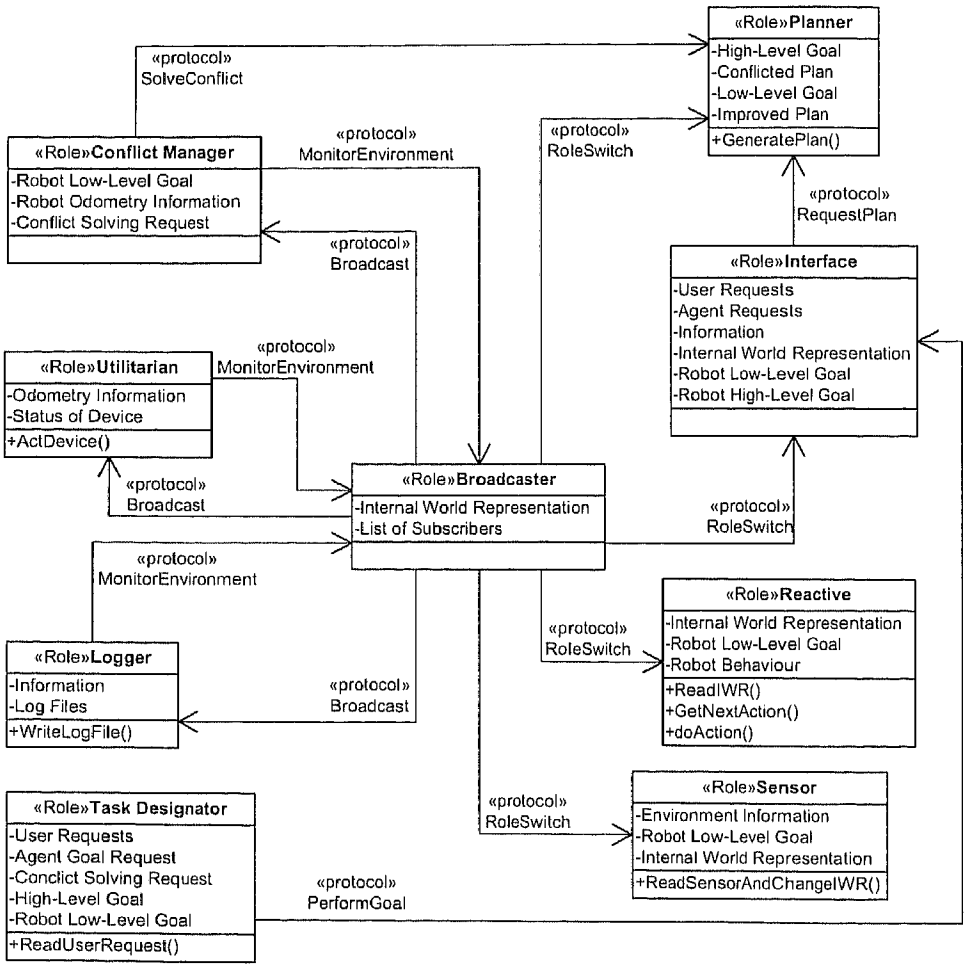


Figure 3.7: Role and Interaction Diagram

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.

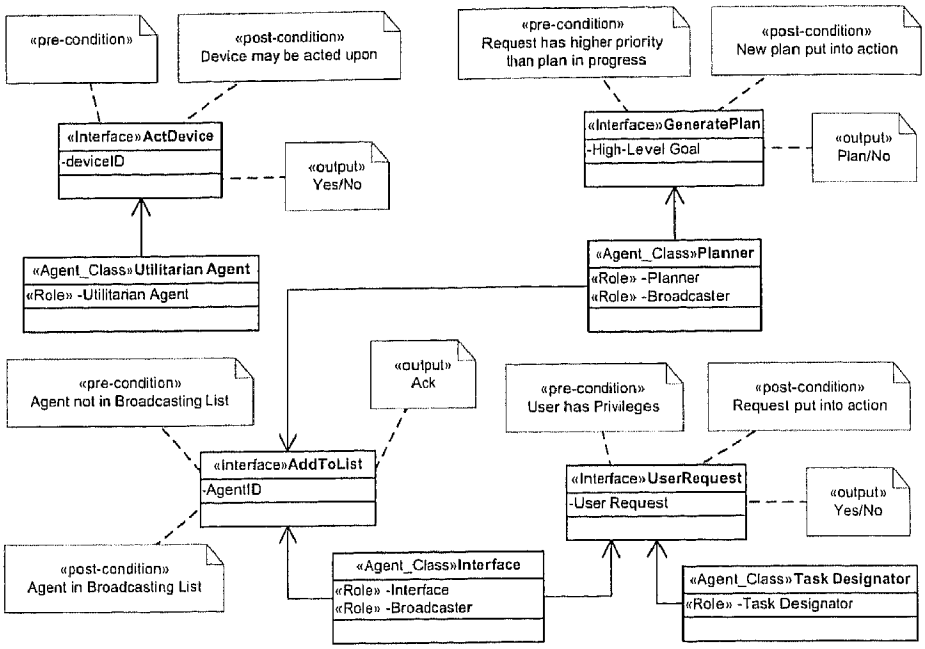


Figure 3.8: Service Model

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:

- 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.

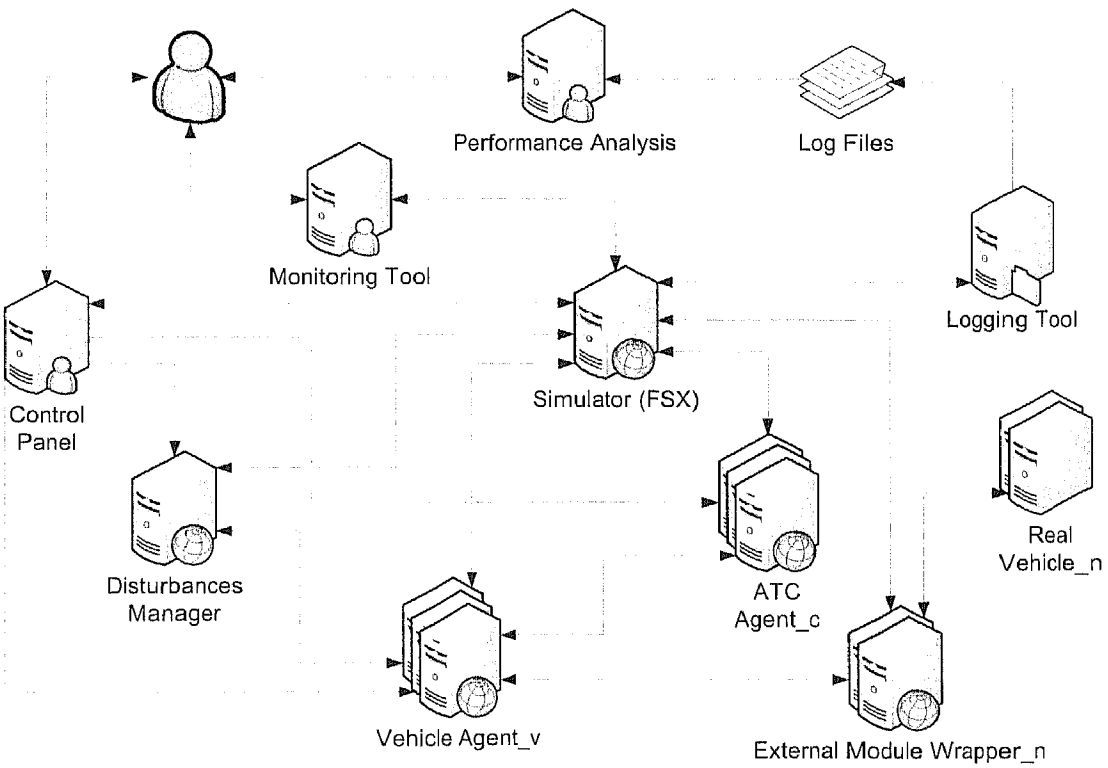


Figure 3.9: General Platform Architecture

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;

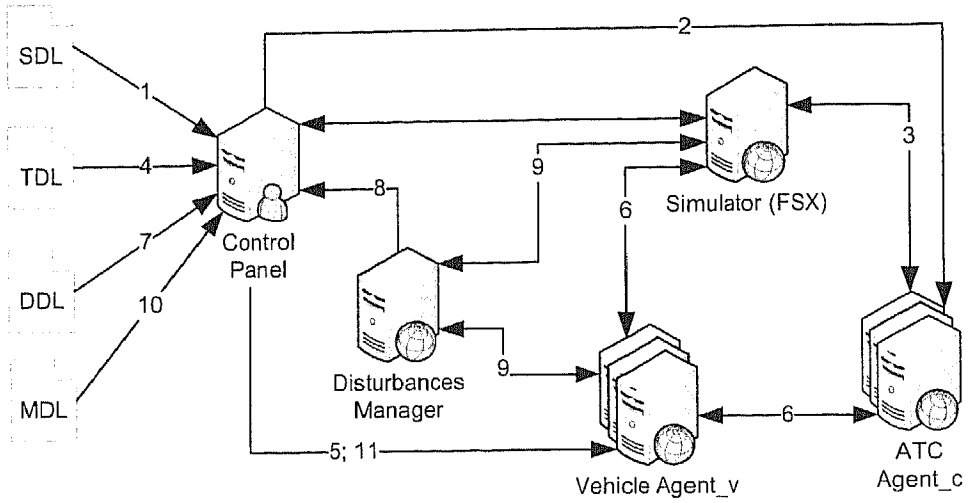


Figure 3.10: Simplified Data Flow Model

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 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.

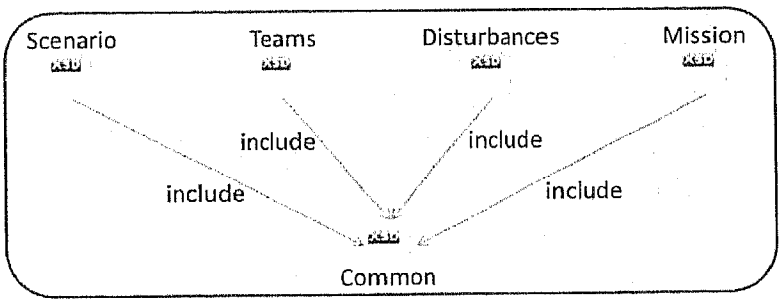


Figure 4.1: Physical Structure of XSD Files for Dialect Specification

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

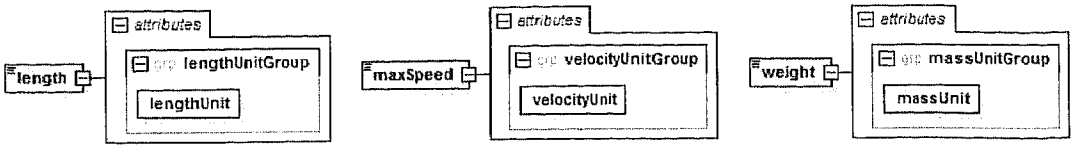
4.1.2 Additional Notes

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

Table 4.2: Measurement Units

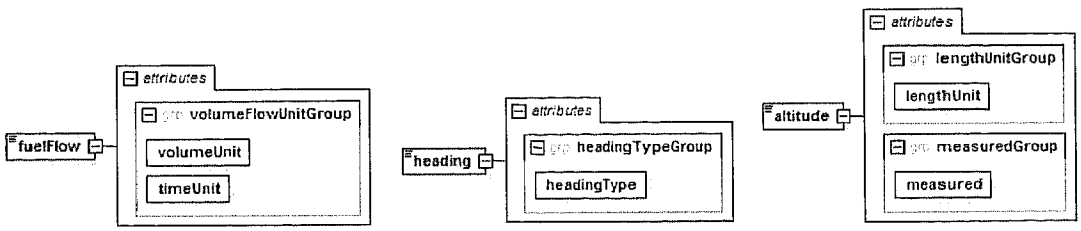
Most of these elements are located in the file that defines the common elements. Figures 4.2(a), 4.2(b) and 4.2(c) show examples of elements that use these attributes, namely the use of length, speed and mass unit attributes. Figure 4.3(a) shows an example of an element that needs two unit attributes,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

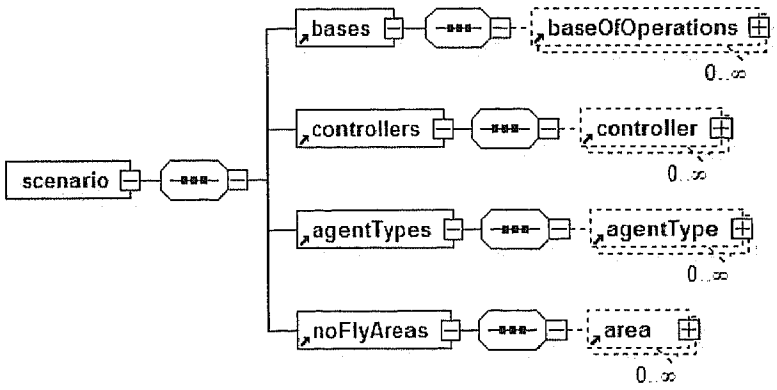


Figure 4.4: Root Scenario Elements

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.

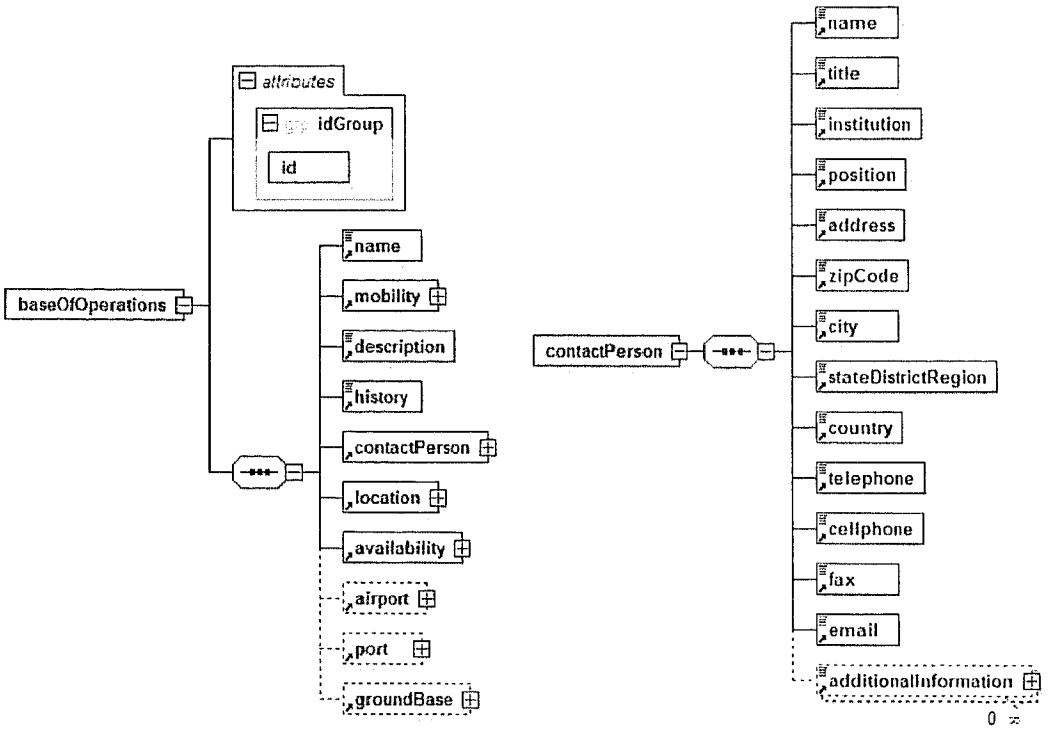
4.2.1 Bases of Operations

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).



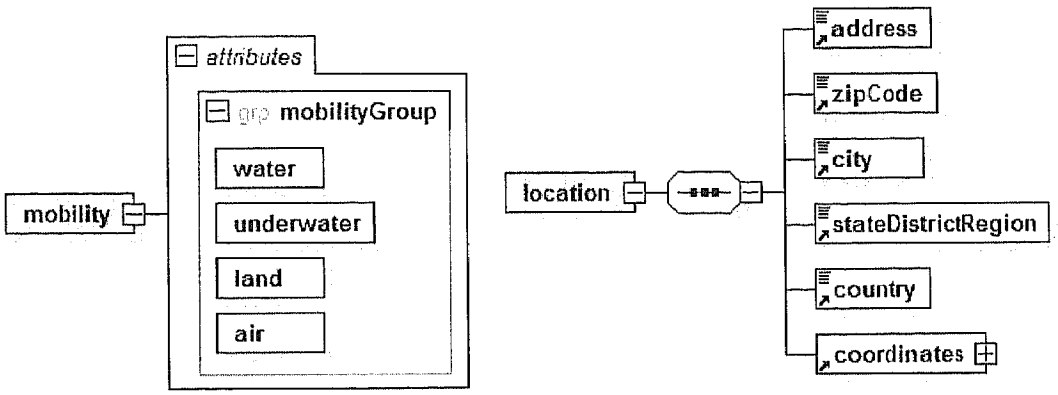
(a) *BaseOfOperations* Element Definition (b) Contact Person

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 night 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 (in which case the *available* attribute would have the value 'always'), 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 (using the *every* and *repeat* attributes, which allow for the recurrence to be defined at a daily, weekly, monthly or yearly basis), 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.

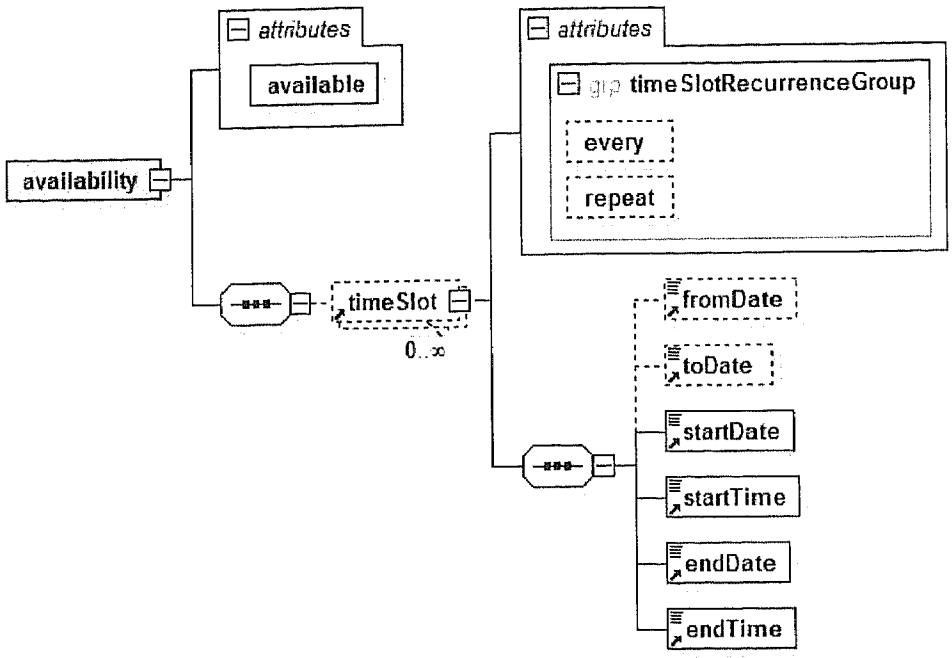


Figure 4.7: Availability Element Definition

Listing 4.1: Availability Element Example

```
...
<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

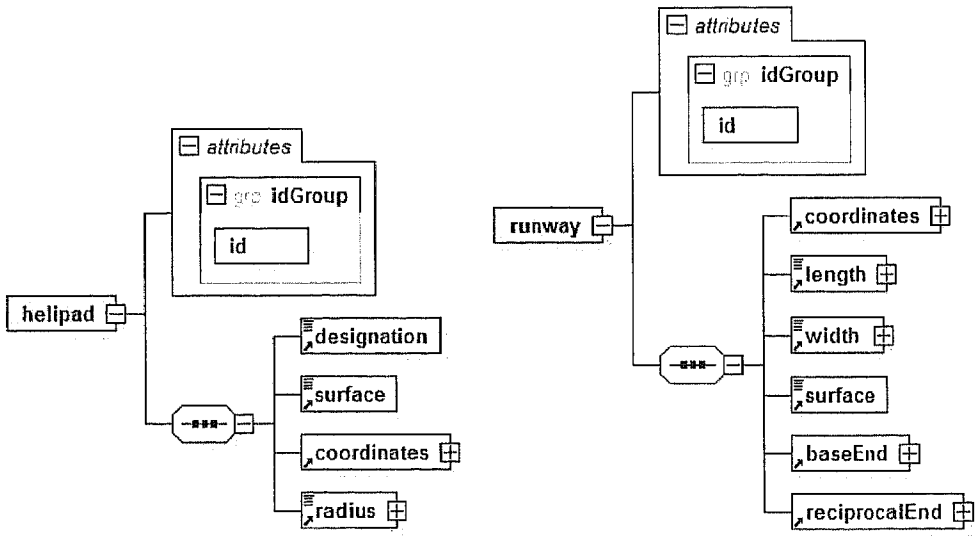
In order to provide a description of airports closer to what is used in the real world, some applications that make use of an airport description were analyzed, as seen in section 2.3. Considering the different formats and information included in the analyzed simulators, it was decided to include a stripped 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.

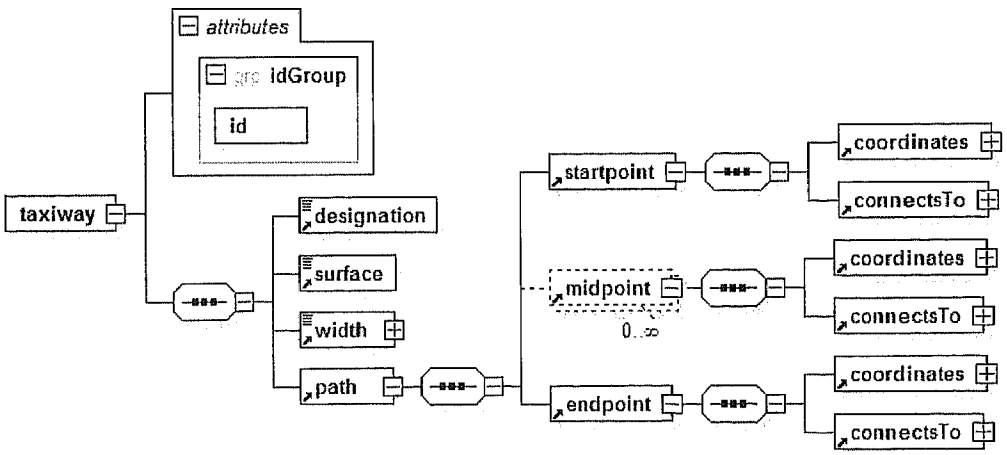


Figure 4.9: Taxiway Element Specification

- **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.

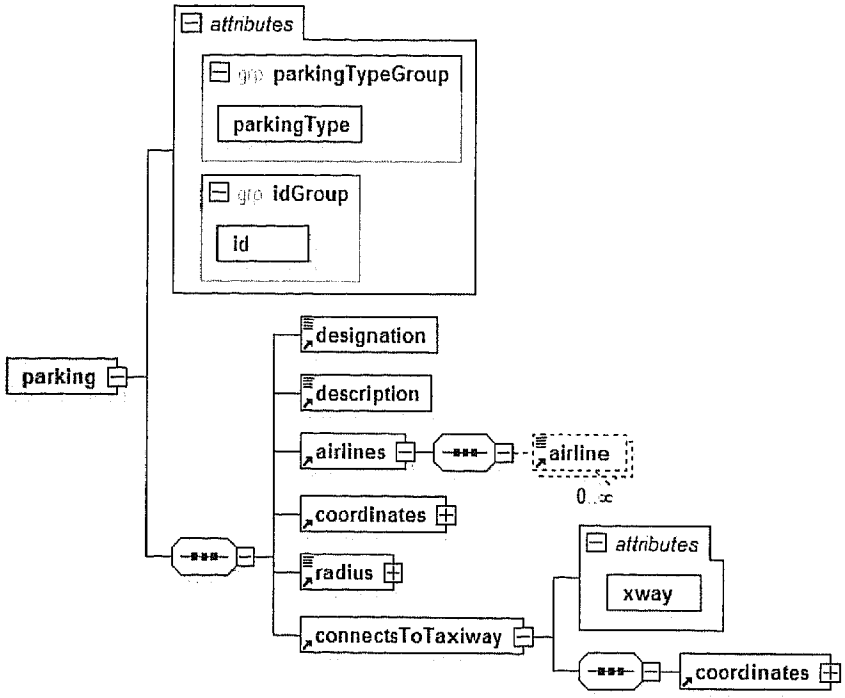


Figure 4.10: Parking Element Specification

- **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.

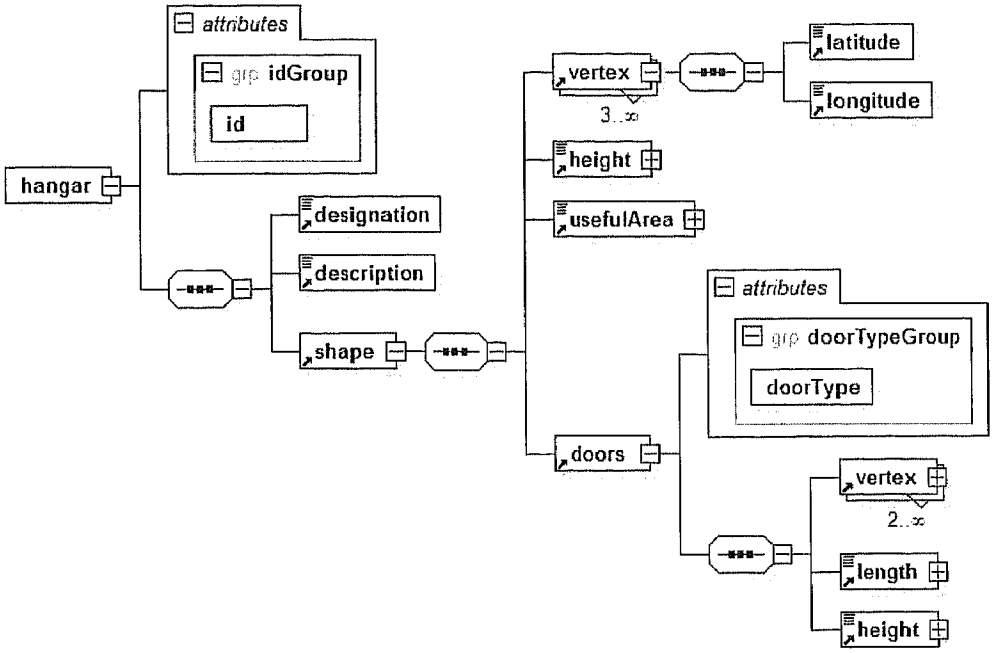


Figure 4.11: Hangar Element Specification

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

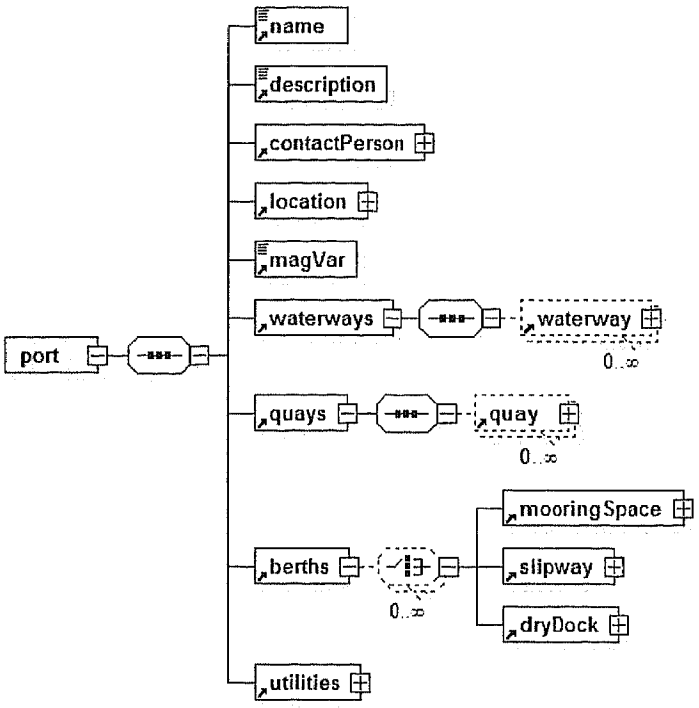
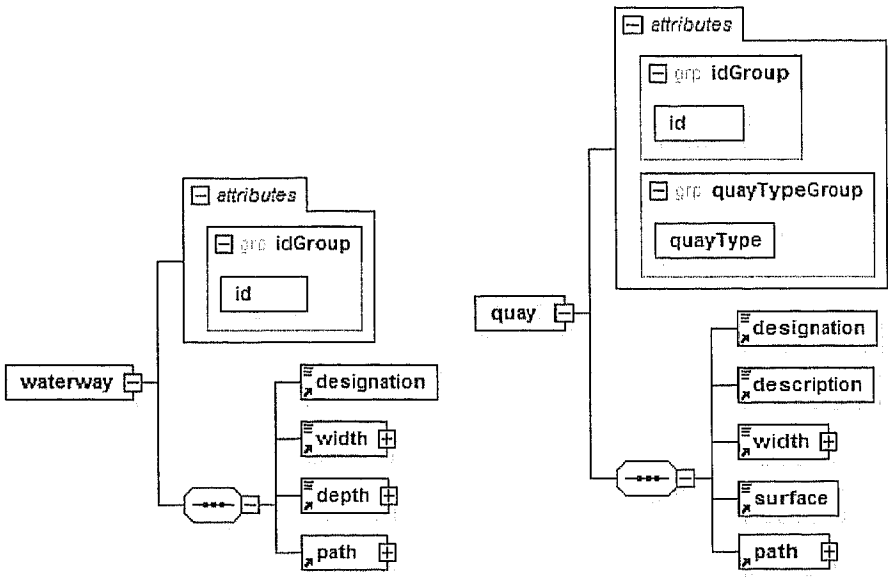


Figure 4.12: Port Element Specification

- **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.