İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

DESIGN OF A DECISION SUPPORT ARCHITECTURE FOR HUMAN OPERATORS IN UAV FLEET C2 APPLICATIONS

M.Sc. Thesis by
Oktay ARSLAN

Department : Defense Technologies

Programme : Information Technologies

JUNE 2009

İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

DESIGN OF A DECISION SUPPORT ARCHITECTURE FOR HUMAN OPERATORS IN UAV FLEET C2 APPLICATIONS

M.Sc. Thesis by
Oktay ARSLAN
514071008

Date of submission : 04 May 2009
Date of defence examination: 05 June 2009

Supervisor (Chairman) : Assist. Prof. Dr. Gökhan İnalhan (ITU)
Members of the Examining Committee : Prof. Dr. Levent Güvenç (ITU)
Assoc. Prof. Dr. Mehmet Turan Söylemez (ITU)

JUNE 2009

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

İHA FİLOSU K2 UYGULAMALARINDAKİ İNSAN OPERATÖRLER İÇİN BİR KARAR DESTEK MİMARİSİNİN TASARIMI

YÜKSEK LİSANS TEZİ
Oktay ARSLAN
514071008

Tezin Enstitüye Verildiği Tarih : 04 Mayıs 2009
Tezin Savunulduğu Tarih : 05 Haziran 2009

Tez Danışmanı : Yard. Doç. Dr. Gökhan İnalhan (İTÜ)
Diğer Jüri Üyeleri : Prof. Dr. Levent Güvenç (İTÜ)
Doç. Dr. Mehmet Turan Söylemez (İTÜ)

HAZİRAN 2009

FOREWORD

I would like to express my deep appreciation and thanks for my advisor Prof. Gökhan İnalhan for his mentorship, sincerity and friendship. I have had great experiences not only related with my research but also about the academic life under his supervision. Therefore, he means much more than an research advisor to me.

I would like to continue by thanking my dear lab mates and it has been a great pleasure for me to being a part of such a great research group. In random order:

Ahmet Çetinkaya for all the great times of friendship, Sertaç Karaman for his recommendation to join such a great laboratory and advice for future plans after graduation, İsmail Bayezit for his companioship during sports activities, Can Kurtuluş for the great coffee and our deep conversations after midnight, Ertan Ümit for sharing his enormous knowledge, and experiences related with non-academical subjects, Melih Fidanoğlu for having such a pure heart, Nazım Kemal Üre for his discussions and chat sessions, Serdar Ateş and Miraç Aksugür for being such a helpful and a warm face.

There is many other names I would like to thank and since it is very hard to say all the names, I am rather preferring to thank them in person.

Finally, I would like to thank my dear family for their continuous support, love and all the other things that they have done for me so far.

May 2009 **Oktay Arslan**

TABLE OF CONTENTS

Page

ABBREVIATIONS

DSS **:** Decision Support System
C2 **:** Command and Control
K2 **:** Komuta Kontrol
UAV **:** Unmanned Aerial Vehicle

LIST OF TABLES

Page

LIST OF FIGURES

Page

DESIGN OF A DECISION SUPPORT ARCHITECTURE FOR HUMAN OPERATORS IN UAV FLEET C2 APPLICATIONS

SUMMARY

UAVs are becoming indispensable assets for military command and control applications such as surveillance, reconnaissance, search and rescue operations because of their superiorities over manned vehicles. Nevertheless, humans are still needed for high-level guidance and commanding of UAVs in those operations for their intelligence and higher level of flexibility in decision making. Within this new position, human operators are responsible for commanding of multiple UAVs under hard timing constraints in a dynamically changing environment. The supervisory control of the UAVs becomes more challenging as the number of the UAVs increases and it is sometimes intractable or infeasible even by a set of operators. In this thesis, we focus on development of decision support tools in order to improve of the agility of a C2 system for UAV fleets and present the framework of a real-time decision support system for operators who are responsible for high-level decision making in scenarios involving a large number of tasks across multiple UAVs. The decision support system consists of three stages including planning, scheduling and low-level mission specific task planning. The overall system is integrated to a labscale multi-vehicle mission simulator, demonstrating the ability of human operators for exploiting a fuller set of UAV fleet capabilities.

İHA FİLOSU K2 UYGULAMALARINDAKİ İNSAN OPERATÖRLER İÇİN BİR KARAR DESTEK MİMARİSİNİN TASARIMI

ÖZET

İnsanlı araçlara karşı sahip oldukları üstünlüklerinden dolayı İHA'lar istihbarat, keşif, arama ve kurtama operasyonları gibi birçok askeri komuta ve kontrol uygulamalarında vazgeçilmez unsurlar hale gelmiştir. Fakat yine de insanlara zekaları ve daha esnek karar verme yetenekleri nedeniyle İHA'ların üst seviyede güdümü ve komutası konularında ihtiyaç duyulmaktadır. İçinde bulunduğu yeni konumda, insanlar dinamik olarak değişen bir çevrede zorlu zaman kısıtları altında çoklu İHA'ların komutası görevini üstlenmektedir. İHA'ların denetlemeli kontrolu İHA sayısı arttıkça gittikçe zorlaşmaktadır ve bazen bu problem birden fazla operatör tarafından bile başedilemez veya yapılamaz hale gelmektedir. Bu tezde, İHA filoları için kullanılan KK sistemlerinin çevikliğini geliştirmek üzere karar destek sistemlerinin geliştirilmesi konusu üzerine odaklandık ve çok büyük sayıda çoklu İHA'ların kontrolu ile ilgili görevler içeren senaryolarda yüksek seviyeli karar vermede görevli olan operatorler için gerçek zamanlı karar destek sistemi çerçevesi sunduk. En basit şekilde karar destek sistemi planlama, iş sırarlama ve alt-seviye görev odaklı iş planlama gibi 3 kısımdan oluşmaktadır. Bölüm sonunda bütün sistem laboratuvar ölçekli bir çoklu-araç görev simülatörüne entegre edilmiştir.

1. INTRODUCTION

The recent developments in the autonomous vehicle technology result in a remarkable increase in usage of Unmanned Air Vehicles (UAVs) in military command and control applications such as surveillance, intelligence or reconnaissance (SIR). Nowadays, UAVs have the capability of performing many complex missions with strict time constraint as well as manned vehicles, even with reduced risks to human pilot. Especially, due to improved human safety, UAVs are, also, preferred in not only military but also civil application such as urban search and rescue operations. However, growing involvement of UAVs in complex applications, several challenging problems arise intrinsically in aerial systems. First, cost of low-level remote control of UAVs might be very high and usually requires employment of several human operators for single UAV of limited decisional autonomy. In addition, following the taxonomy as presented in [1], the number and types of missions generally require several UAVs operating collaboratively in order to accomplish the operational goals. Therefore, new tools, algorithms and architectures are required to maintain coordinated control of UAV fleets.

Despite the current trend reducing the role of human in military systems and substantial interest focused on the development of higher level of autonomy for UAVs, human operators are still needed for supervisory control of UAV fleets. As proposed in [2], one way to exploit UAVs' capabilities is employing the human operators for higher levels of planning and decision-making tasks rather than direct manual control of UAVs. Since human operators have limited capacity of cognitive resources, there has been a great deal of effort in development of systems that allow one operator to manage of a large number autonomous UAVs in a high mission workload environment. While there are several successful works that encourage the controlling of multiple UAVs by a single operator [3], the supervisory control of the UAVs becomes more challenging as the number of the UAVs increases and it is sometimes intractable or infeasible even by a set of operators due to high degree of mental workload.

Therefore, new C2 systems (human operators with their supporting information systems and decision aid tools) are needed to be developed in order to increase the agility of the overall system including human operators in means of responsiveness [4]. There are various factors that effect the cognition of the operators, but one of the most important and time consuming process is deliberation of the which tasks to perform and scheduling of these tasks regarding the temporal and resource constraints. Since optimization of human interaction with an automated system through supervision is very difficult, one way to mitigate saturation of operators is providing real-time decision aid for planning and scheduling of these tasks. Such a decision support system contributes too much to the human operators to command and control of the UAV fleets effectively in a timely manner. Because in many of the proposed systems [5], [6] the duration of the planning and scheduling of the tasks is ignored and there are assumptions that planning and scheduling are instantaneous. However, these processes take time in the case in real applications and cause to saturate the cognition of human operators and therefore they have to be handled as fast as possible with the help of real-time generated decision aids.

In this work, we are interested in development of more agile C2 system for UAV fleets by developing decision support tools and present the framework of a real-time decision support system for operators who are responsible for high-level decision making in scenarios involving a large number of tasks across multiple UAVs. Then, the overall system is integrated to a labscale multi-vehicle mission simulator, demonstrating the ability of human operators for exploiting a fuller set of UAV fleet capabilities.

1.1 Organization of the Thesis

This thesis is mainly divided in two different parts. In the first part, decision support systems are examined and the proposed decision support architecture and its main segments are described in detail. On the contrary, the second part is focused on more technical works and the general design and the architecture of the multi vehicle mission simulator is presented in detail the next section. Finally, the integration of the decision support architecture to the simulator and its implementation are given and some successful results of the scheduling algorithm is also given.

1.2 Contributions

The main contributions of this thesis can be briefly summarized as follows:

- We provide the design and the development of a multi-vehicle mission simulator structured for joint real-time simulation across manned-unmanned fleets, and the mission control center. The hardware structure within the network simulator is tailored to mimic the distributed nature of each of the vehicle's processors and communication modules. Open-source flight simulation software, FlightGear, is modified for networked operations and it is used as the 3D visualization element for the pilot and the mission controls.
- We focus on development of decision support tools in order to improve of the agility of a C2 system for UAV fleets and present the framework of a real-time decision support system for operators who are responsible for high-level decision making in scenarios involving a large number of tasks across multiple UAVs. The overall system is integrated to a labscale multi-vehicle mission simulator, demonstrating the ability of human operators for exploiting a fuller set of UAV fleet capabilities.

2. DESIGN OF A DECISION SUPPORT ARCHITECTURE

The problem of planning and scheduling of tasks can be solved by human operator easily in the recent supervisory control architectures that based on simultaneous control of four or five UAVs by a single operator. However, as the number of the tasks and UAVs increases like order of hundreds, this problem becomes very challenging and time-consuming and it is required to design a higher level layer which is solely responsible for deciding which actions to perform and temporal allocations of them.

The decision support architecture that is envisioned to provide real-time decision aid to the manned vehicle or ground operators is embedded over the UAV architectures given in [7], [8] and it consists of four main segments, namely, planner, scheduler, resource and task manager as it is shown in Figure 2.1. With including of this new highest layer, there is an operator who is mainly responsible for deliberation of tasks and temporal and resource allocation for them. In addition, the set of human operators who are responsible for supervisory control in the underlying UAV control architecture defines a new type of resources in this extended architecture. After defining resource and time consistent tasks with the aid of the decision support system, this operator delegates or assigns them to the available supervisory operator and allocated UAVs via human machine interfaces.

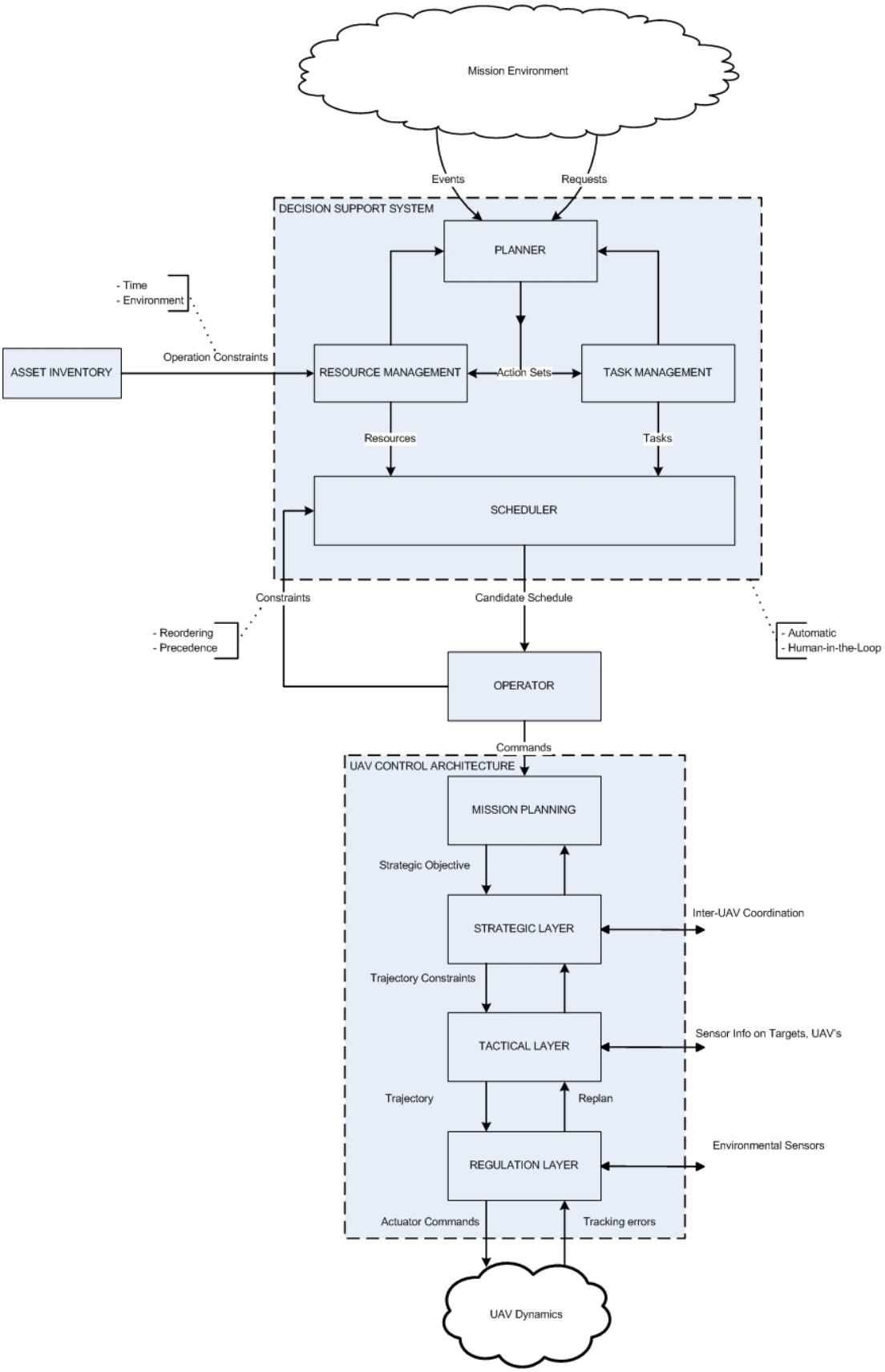

Figure 2.1 : Event-Driven Decision Support Architecture

2.1 Planner

The planning is the problem of deciding which action sets to perform during the scenario. Planner is triggered by an external monitored event such as bombing, threat detection or a task request from mission control center. Following the taxonomy as presented in [1], the planning step proceeds by selecting suitable action from a predefined action sets by an expert system. The planner continues its execution collaboratively with resource and task manager until there exists consistent resource allocation and set of tasks.

- **Events/Requests** The set of phenomenon in the mission environment corresponds to the events and the set of commands from mission control center or other peers correspond to the requests. Each event and request has a different level of priority and importance and requires different type of actions to be selected by the planner.
- **Action sets** The actions are the basic components of missions like loitering at a specific point or returning to the base. Different types of UAVs in the inventory can perform each mission. Each mission has different goals and functional/information requirements and requires different number of UAVs to be performed.

2.2 Integration of Planner and Scheduler

- **Resource manager** Since each task requires different type and number of UAVs; it is required to allocate enough number of UAVs for each required type. Therefore, resource manager is responsible for resource allocation satisfying time and environment constraints in the asset inventory. However, these tasks are not necessarily mutually exclusive, one type of UAV may have the capability of performing multiple tasks, and there may be several different types of allocations for the same problem. Therefore, resource allocation must be done wisely and the solution must be selected from the set of allocation by considering further requests.

- **Task manager** This manager is responsible for defining specification of each task by examining the action sets found by planner.

2.3 Scheduler

The scheduling is the problem of temporal allocation of resources to the tasks. Before the scheduling process, a set of resources must be allocated regarding operation constraints (time, environment) and a group of tasks must be defined regarding to the selected actions and resources. Then, the scheduler finds a candidate schedule for these tasks under resource constraints by satisfying specified time windows constraints. The human operator has the right of the modifying the candidate solution or rescheduling the problem with new precedence and temporal constraints. This scheduling process is executed until the operator confirms and submits the candidate solution as a mission commands to the allocated UAVs and corresponding supervisor operator.

- **Operation Constraints** Due to complex nature of the missions and resources, a scenario intrinsically contains different types of constraints such as time, precedence. Structural dependencies between missions, physical and logistic constraints define a set of temporal constraints. For example, a target must be destroyed (attack) within some periods of time immediately after its designation (reconnaissance), the total completion time of set of missions assigned to a UAV cannot exceed its endurance and it must return to base for maintenance before a certain amount of time from endurance.

As classified in [1], each activity (UAV missions) has different characteristics based on its goals, functional/information requirements. For instance, the target acquisition mission requires path planning (areas to search and waypoints to the area of interest), threat area and forbidden zone information or cargo mission requires path planning (route from origin to destination), forbidden zone information and scheduling mechanism. These kind of planning problems are solved in the underlying mission planning layer of UAV architecture.

3. DEVELOPMENT OF A MULTI-VEHICLE MISSION SIMULATOR

Due to recent advances in science and technologies, many researchers attract applications of UAVs in both civilian and military areas. The capabilities of UAVs in many applications are much more than conventional manned vehicles, since they can offer longer mission time, high endurance, performing hazardous mission. In order to benefit all these growing capabilities of UAVs, it is essential to develop new control structures and algorithms. However, because of the increasing complexities in missions and safety-critical requirements in avionics, all control and coordination algorithms, avionics hardwares and vehicles must be tested on a realistic lab-scale testbed before using them in the real mission scenarios.

There are too many developed testbeds for testing cooperative control algorithm from other universities all around the world. For instance, at MIT, there is a Multi-vehicle Testbed in the Aerospace Control Laboratory and it has several capabilities such as testing distributed command and control algorithms, performing multi-vehicle operations with humans-in-the-loop and researching on vehicle/fleet autonomy. The vehicle fleet of testbed consists of several vehicles from four classes (UAV's, helicopters, blimps, rovers) and these various vehicles increase the broad range of applications.

Another well-known testbed is Bear, Berkeley Aerobot Team, in University of California. BEAR research testbed include a fleet of rotor-wing UAVs, and fixed-wing UAVs, unmanned ground robots, and a mobile ground station. This high-tech testbed provides to the researchers the opportunity of studying on the development of advanced theories and approaches for multi-agent coordination, vehicle guidance and control, and other relevant fields that require aerial platforms such as dynamic wireless networking and vision-based navigation.

The rest of this work is organized as follows:

In Section 2, we first introduce the problem of distributed command and control (DC2) by examining it from system-theoretic view of point. Then, the problem is

analyzed and defined as a set of functional requirements by using Use Case Diagrams that is a well-known approach in software engineering. In Section 3, the communication requirements in this vehicle network are examined as a sub problem of DC2. Then, in the Section 4, the developed simulator architecture is given in detail by explaining all its component and the software architecture of Communication and Information Distribution Module and several test cases are mentioned in the next section. Finally, we conclude by explaining about the current researches going on the simulator and future works in the end of paper.

3.1 The Problem of Distribute Command and Control

3.1.1 Problem Description

The Distributed Command and Control (DC2) problem can be modelled from system-theoretic point of view, since there are a set of objects, a common goal need to be accomplished by these objects, and finally a set of functionalities that the system should have. In addition, this problem can be considered as a higher-level problem based on Distributed Information Gathering problem [9]. A conceptual architecture can be designed by analyzing the functionality requirements and how this functionality should be achieved.

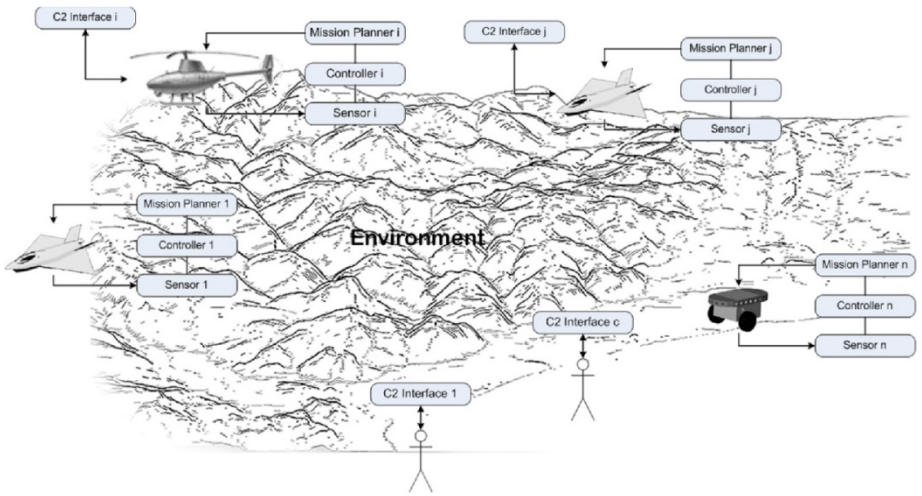

Figure 3.1 : Distributed Command and Control Problem

The Figure 3.1 is an overview of the DC2 problem and it shows the environment where the all mission is performed, human operators, and unmanned-manned vehicles. All entities, both humans and vehicles, can be considered as members of cooperative teams in the fleet working towards accomplishing a common goal. From

software engineering point of view, these entities correspond to software objects, such as, human operators who are co-pilots or commanders in the mission control center are represented by Command and Control Interface (C2I), embedded computers, and avionic hardwares are represented by Mission Planner, Controller, and Sensor.

There is a set of heterogeneous objects, which are equipped with sensors, actuators, and have the capability of communication with other objects. These objects are located in an environment where contains a distributed dynamic phenomenon and perform a specific mission collaboratively. This phenomenon may be sensor data about the environment to be measured, or just a suddenly existing events to be monitored, reported. Each object can contribute information to the vehicle network by exchanging its local information through a communication channel. The term object has been used as the same as vehicle "which has specific mission related tasks, purposes" so far and it is also synonymous with a decision maker or an agent.

In addition, there is a set of human operators interacting with the vehicle network through Command and Control Interfaces (C2I). Operators are the people who are co-pilots in a manned-vehicle or commanders in the mission control center. They can request information about environment from vehicle network, assign tasks to a specific vehicle or enter information about the environment by observing it directly using human senses (reporting events existing suddenly).

All entities, human and vehicle, form a cooperative team and work collaboratively to achieve an objective of a specific mission. E.g. pursuit-evasion, reconnaissance, surveillance, search and rescue.

This section examines the DC2 problem and proposes network architecture by using software engineering approaches, such as analyzing functional requirements of system.

3.1.2 Functional Requirement

There are too many approaches used for specifying functional requirements of a system and UML Use Case diagram, a common approach in software engineering, is used for analyzing functional requirements. The world is modelled as compound of three subparts: the cooperative unmanned-manned vehicle network (CUMVN), the environment which contains the phenomenon, and the rest of the world in the Figure

3.2. The operation of CUMVN can be grouped in three primary categories: collection and distribution of information, commanding and controlling of the network by operator, and planning and coordination of the mission. There are two main actors in the Use Case diagram, who are operator and the phenomenon. Operators may gather information about environment by both interacting with vehicle network and observing it directly using human senses such as seeing or hearing an event. Finally, the network just observes the phenomenon in the environment without interacting with it via sensors belong to each member vehicle and all gathered information are fused and circulates over the network.

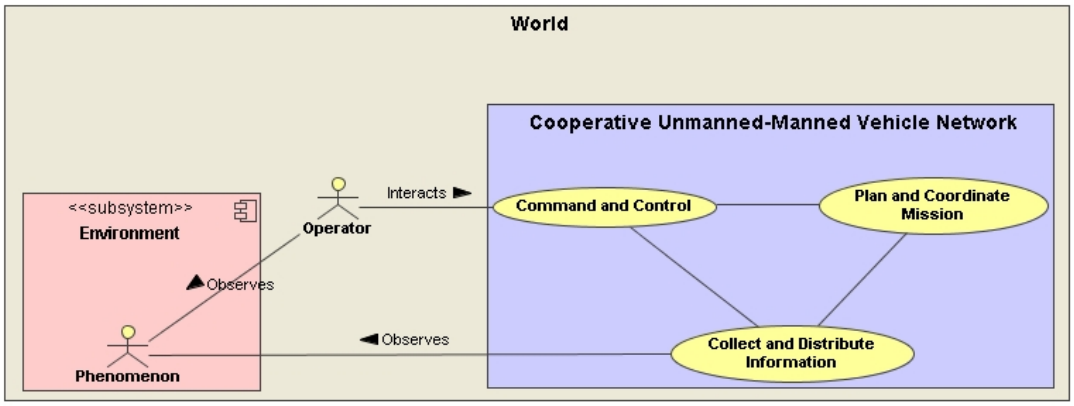

Figure 3.2 : Level 1 use case diagram for DC2 problem

Each use cases are examined deeper in the Level 2 use case diagram which is shown in the Figure 3.3. There are two main types of operations, decision-making and cooperation, and information gathering and distribution, respectively. Three main use cases have several included sub-cases which have tasks related to decision-making or information gathering. These three main use cases are divided into sub-tasks with the UML << include >> associations. The Command and Control use case includes all the functions done by human operator in order to access the vehicle network. It has five sub-cases and these interactions are related both decision making and information distribution. Operator can perform functions in the Request and Send/Receive Information and Enter Information use cases which are the primary operation for human operators. Operator enters their information related requests by using a pre-defined communication protocol to the vehicle network and the network responds with related information by using the same protocol.

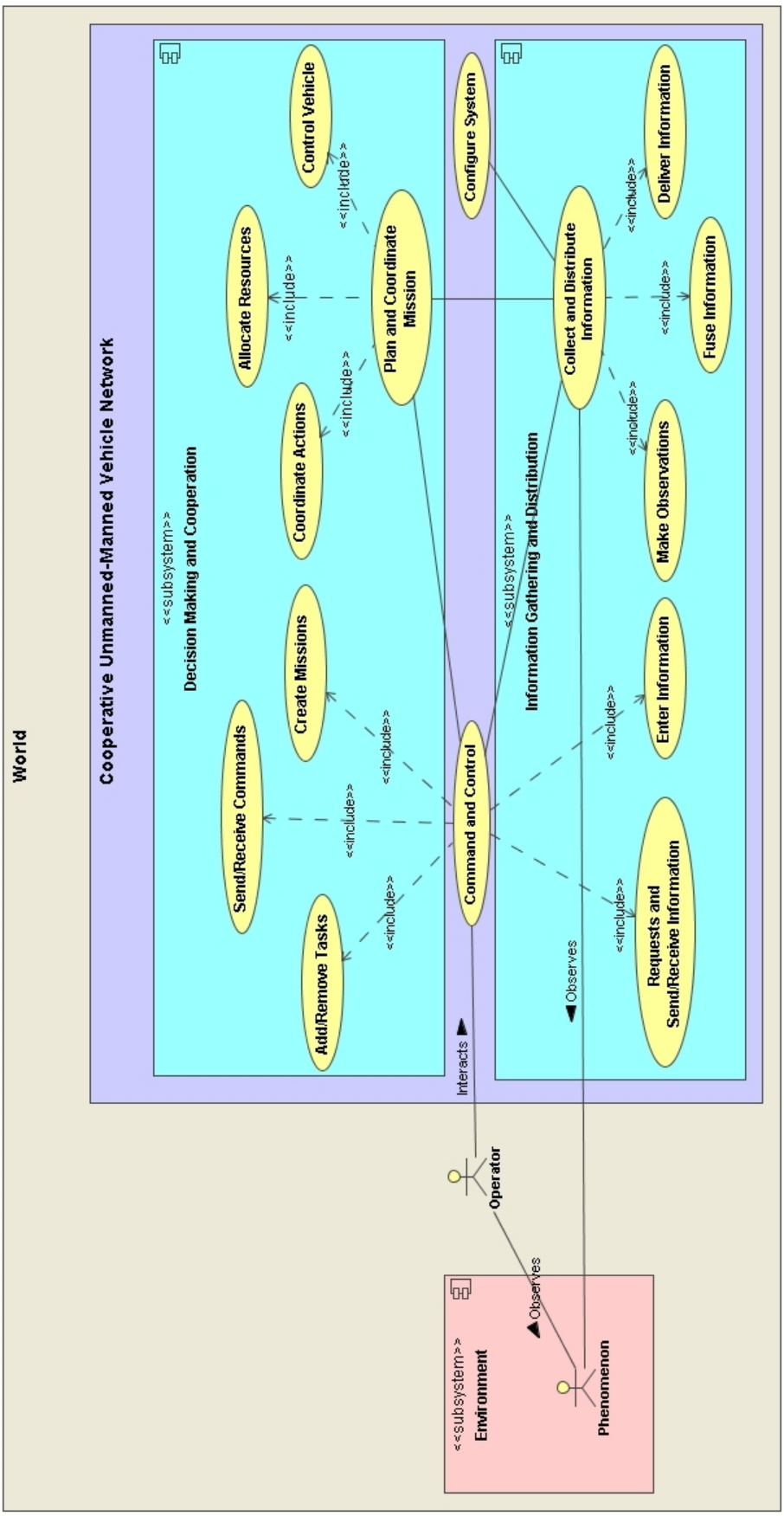

Figure 3.3 : Level 2 use case diagram for the DC2 problem

The Collect and Distribute Information use case is the core one in the Information Gathering and Distribution layer. It is very crucial for all members of the vehicle network to have global information, usually derived from combining the data obtained from local different sources, about mission. Finally, the information must be distributed to the destination addresses simultaneously as information requests are taken.

The Plan and Coordinate Mission use case includes decisions and planning about mission at highest level of view. Actions related with mission which are either for an individual vehicle or a group vehicle are decided based global information taken from other vehicles and human operator requests entered via C2I. Hence, this use case has close interaction with both of two main use cases, previously described.

3.2 Communication Problem

3.2.1 Problem Description

Communication and data distribution in cooperative unmanned-manned vehicle networks is a challenging problem due to mission-driven complex information flow requirements. In addition, while achieving fleet-level coordination, the unmanned-manned airborne fleet network must also remain in communication with other non-fleet entities such as commander units as well as ground stations, information nodes which forward requests and enhance situational awareness. Because of such needs, data exchange across not only similar units, but also across dissimilar units (ground stations-vehicles) becomes a critical factor.

Figure 3.4 illustrates the cooperative unmanned-manned vehicle networks from communication and information distribution perspective. It shows vehicles as nodes, embedded computers and sensors as local information sources, and communication modules. Each entity such as unmanned-manned vehicle or ground assets in the network has several local information sources such as sensors, embedded-computers or C2 interfaces. This information may be purely sensory data about environment or states of the vehicle need to be contributed to the network for fusing operations, or it may be commands, mission related requests that need to be delivered to an individual vehicle or a group of vehicle in the network in order to achieve the predefined mission collaboratively. Because of such mission-driven information flow

requirements, a standalone module should handle communication with other entities and information distribution in the overall network.

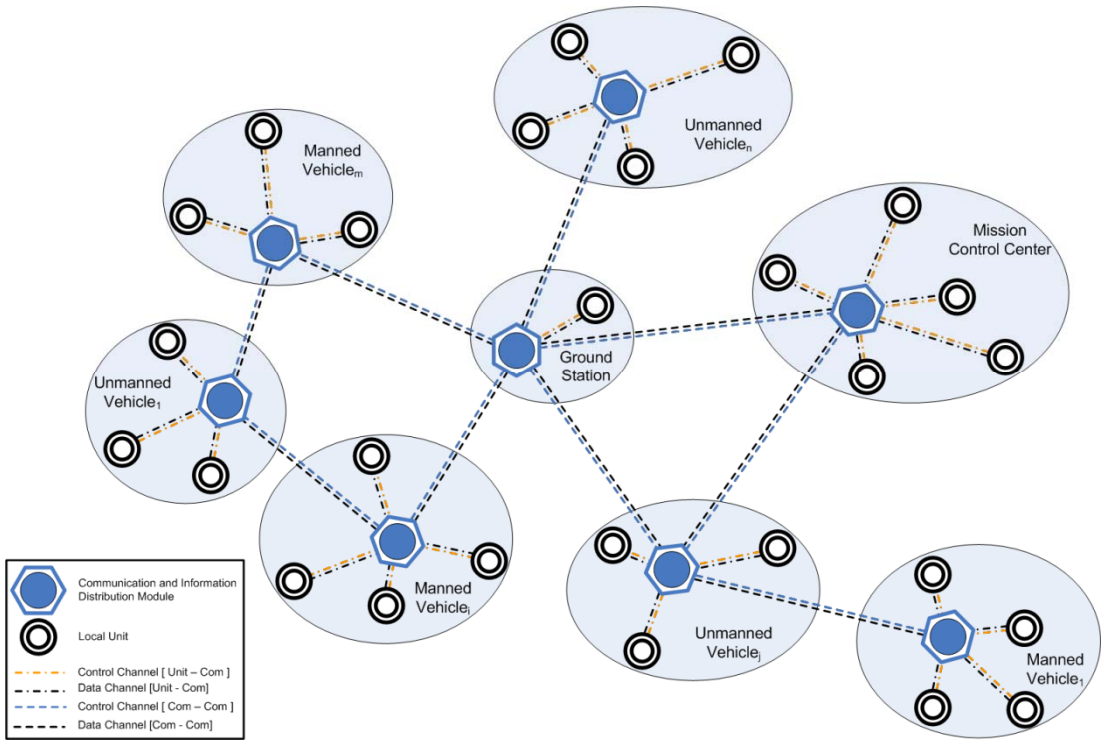

Figure 3.4 : A conceptual view of Communication and Information Distribution Problem

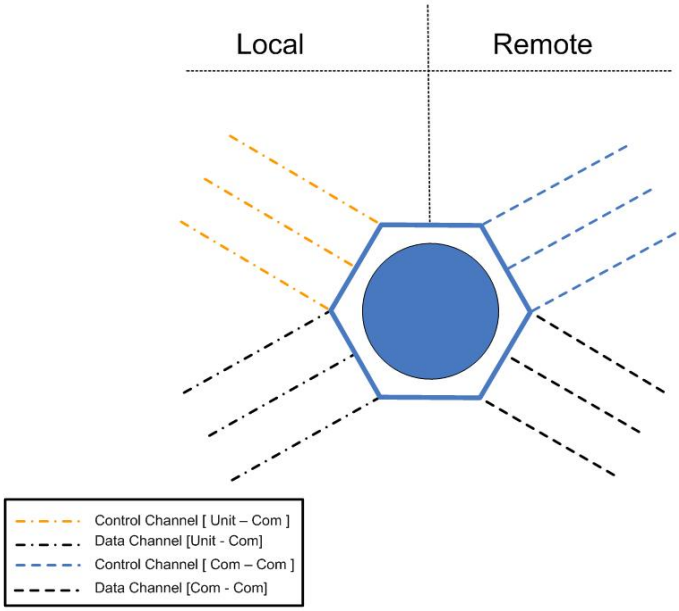

Figure 3.5 : Conceptual design of the Communication and Information Distribution Module

Figure 3.5 shows the conceptual structure of a single CID module. Each local unit of the vehicle and remote CID module of others' vehicle are connected to the module with both data and control channels. Data about environment, states, or other specific information are gathered from local units and distributed to specific destinations via data channels. In addition, routing of information flow is changed according to the commands, requests and special messages taken via control channels.

Actually, what the CID module's objective is gathering control messages which are sent by other entities in the network from control channels, and serving for requests and commands according to the control messages such as changing the information flow in the data channels.

3.2.2 Functional Requirements

The operation of CID module can be separated into four groups: configuring system, controlling, forwarding and transporting information. There are three main actors in the Use Case diagram, who are configuration data, control packet, and data packet.

First, the CID module has to be configured according to special configuration data including name of communication interfaces, parameters, address of units and other CID modules in the network etc, so that Configure System use case handles these operations when the module turns on.

The CID module operates in two different planes: control plane, and forward plane. Parsing of control messages taken via control channels and controlling of the data channels are handled in the control plane. Forwarding plane is responsible for the continuous or periodic routing of the data taken via data channel. These two planes heavily use low-level functions which are implemented Transport Information use case and perform the actual process of sending and receiving a packet received on a logical interface to an outbound logical interface.

Each use cases are expanded into several subtasks in the Level 2 Use Case diagram which is shown in Figure 3.7. The Configure System use case includes all operations related with initialization of connection tables, creating of controller objects and all control communication interfaces.

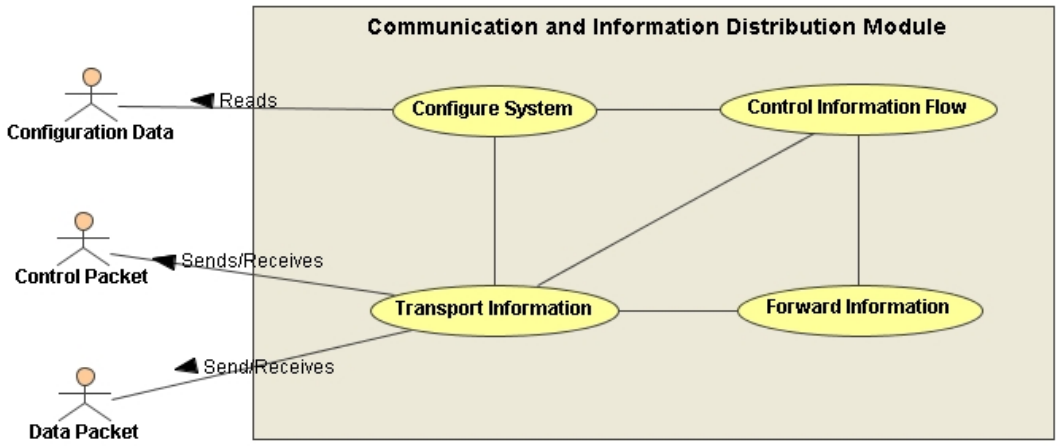

Figure 3.6 : Level 1 use case diagram for Communication and Information Distribution Module

Control Information use case handles operation related with parsing of control messages. Actually, there is one controller object running in this plane and it manages preparing of the data communication interface object and forwarder objects according to control messages. It can control the information flow by updating routing tables of forwarder objects.

Forwarding Information use case has several subtasks related with updating of routing table, periodic and continuous routing operations. There may be some data routing requests from other clients and controller object can updates the routing table by using routines of forwarding plane. Routing of information may be requested as continuously or periodically. Therefore, there are two different use cases for continuous and periodic routing operations, separately.

Transport Information use case includes operations about configuration of several physical communication interfaces such as CAN, Ethernet, Serial etc. Low level sending and receiving operations are implemented by functions implemented in this use case.

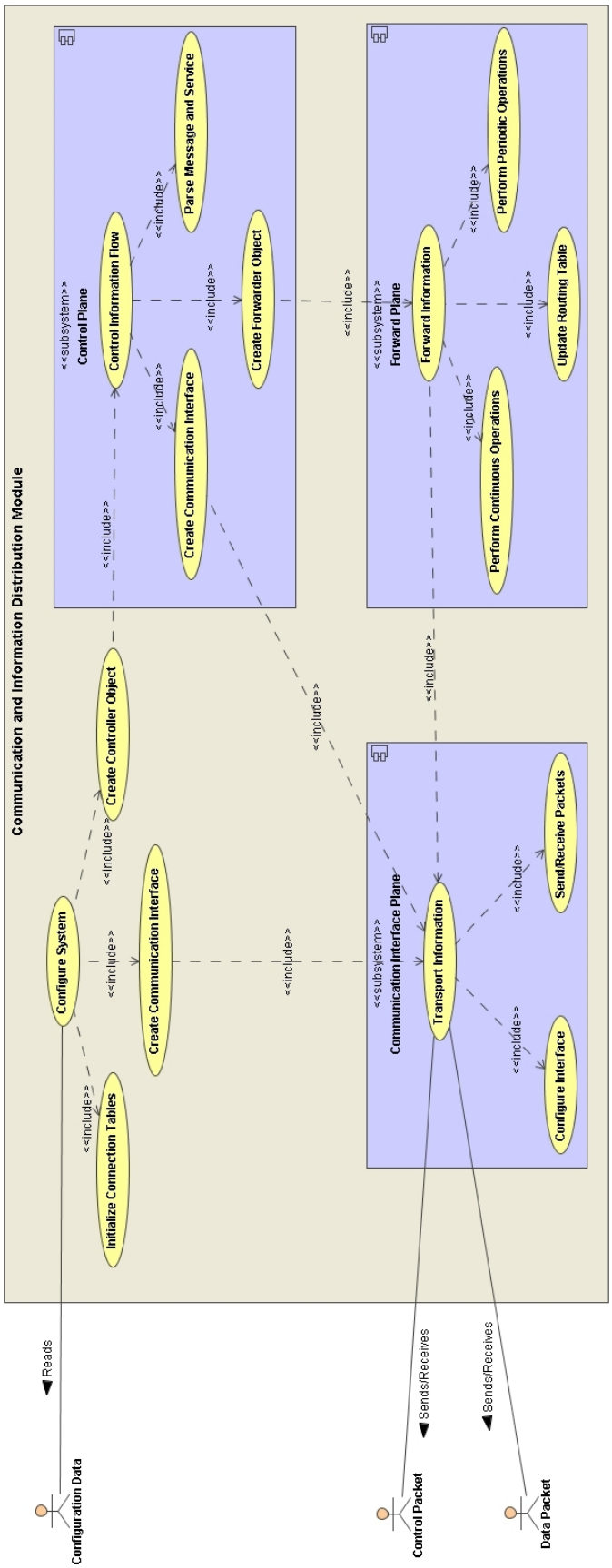

Figure 3.7 : Level 2 use case diagram for Communication and Information Distribution Module

3.3 General Design and Architecture of the Simulator

An experimental network simulator is developed for joint real-time simulation across manned-unmanned fleets, and the mission control center. The hardware structure within the network simulator is tailored to mimic the distributed nature of each of the vehicle's processors and communication modules. Open-source flight simulation software, FlightGear, is modified for networked operations and it is used as the 3D visualization element for the pilot and the mission controls. The UAV dynamics and low-level control algorithms are embedded within the xPC target rack. Equipped with 3D flight simulation displays and touch-screen C2 interface at the desktop pilot level, the platform also allows us to rapidly prototype and test pilot-unmanned fleet supervisory control and pursuit-evasion game designs [10]. Figure 3.8 illustrates the general architecture of the simulator.

3.3.1 Simulation Control Center

Operator Panel: It is used for configuring the network setting of the simulator. All information about the simulation (numbers of manned-unmanned computers, IP addresses of the machines) are inputted from this GUI.

World Computer: This computer is used for controlling of the packet traffic of the visualization layer. Several programs which are called State Sender run on this computer and make packet conversion from Dynamic (D) packet to the Multiplayer (M) or Flight Dynamic Model (FDM) packets which are needed for visualization of the simulation. All the unmanned-manned xPC computers in the simulator send their states as Dynamic (D) packet to these programs at a specific frequency and State Sender programs convert the dynamic (D) packet to the multiplayer (P) packet and send them to the both FG_Server and 3D Multi-monitor visualization systems of manned simulator. In addition, the State Sender programs for the manned simulators make a dynamic (D) to Flight Dynamic Model (FDM) packet conversion and send the FDM packets to the 3D Multi-monitor visualization system for cockpit visualization of the manned simulator.

State Sender, being one of the unique parts of network simulator, basically is a piece of C++ program that listens dynamic states information of a vehicle or any other simulation object and when it receives such information it makes the packet conversion.

The problem with multiple display systems is that since the dynamics of the aircraft belongs to the main computer, slave (left/right) computers should not connect to FG Server. Any connection from left/right servers will lead to strange behavior of visualization happening because of delays between packet transfers. Without any connection to FG Server left/right computers would not be aware of any other simulation objects (eg. other vehicles), and they would not be able to visualize such objects. State Sender also sends the constructed packets to LEFT and RIGHT computers to solve this problem. Yet, this is the preferred solution of simulator for the "multi-monitor multiplayer problem".

3.3.2 Virtual Unmanned Vehicle Simulation

Virtual unmanned simulation has 4 components: mission coordination computer (MCC), xPC computer/controller, xPC computer/dynamics, and CID module. There are two types of information flow in the network, namely, mission information and visualization information. Because of different types of information, there are two different communication bus, CAN and Ethernet bus, which are all components are connected. Can bus is used for mission related information flow, while Ethernet is used for data flow of visualization system. The components of the VUV simulation are shown in Figure 3.9.

Mission Coordination Computer: It is used for coordination between other manned-unmanned vehicles in the fleet when performing a mission collaboratively. Higher level planning and coordination algorithms usually run on these computers in order to achieve specific tasks collaboratively. This program has manly three sub modules; communication module, flight mission module, and task modules. Flight Mission Module is responsible from controlling of data flow in MCC and between sub modules. CID module is used for communication between other MCCs, and task modules may be any special algorithms like task assignment, pursuit evasion.

xPC Computer/Controller: This computer is used to implement the controller algorithm of the unmanned vehicle. Due to its fast development and rich built-in blocks, Matlab/Simulink is chosen for development of the controller algorithm.

xPC Computer /Dynamics: In order to simulate the vehicle dynamics in almost real-time, Matlab/Simulink models xPC Target technology is used. Matlab/Simulink simulation models are compiled to native code to run within xPC operating system

on a target machine. After configuring the settings of xPC Target, a diskette including configured operating system is prepared. An ordinary computer can be used as an xPC Target computer by booting it with this diskette. An appropriate ethernet configuration enables connecting and uploading the Simulink model to the xPC Target computer. After uploading the model, simulation can be started using the remote control via ethernet. xPC Target configuration screen brings an option for executing the uploaded model.

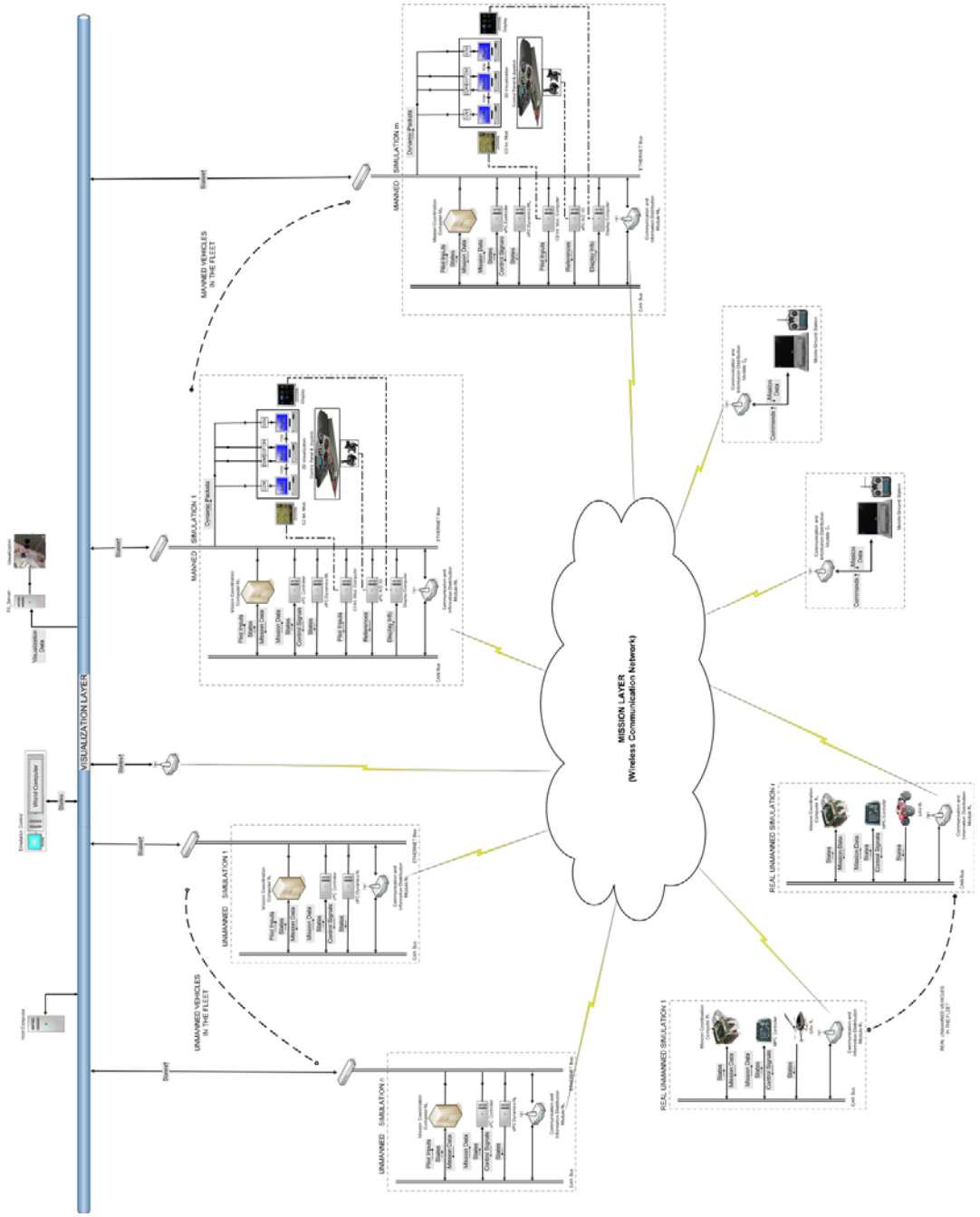

Figure 3.8 : General architecture of Cooperative Manned-Unmanned Network Simulator

CID Module: Each individual component has to use CID module in order to join the CUMV network. This CID module implements a simple communication protocol that is used among components of the network for communication and information distribution.

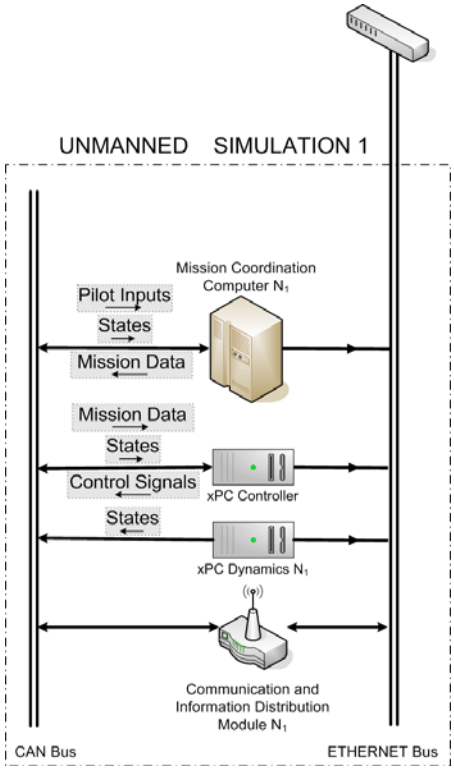

Figure 3.9 : Components of the Virtual Unmanned Vehicle Simulation

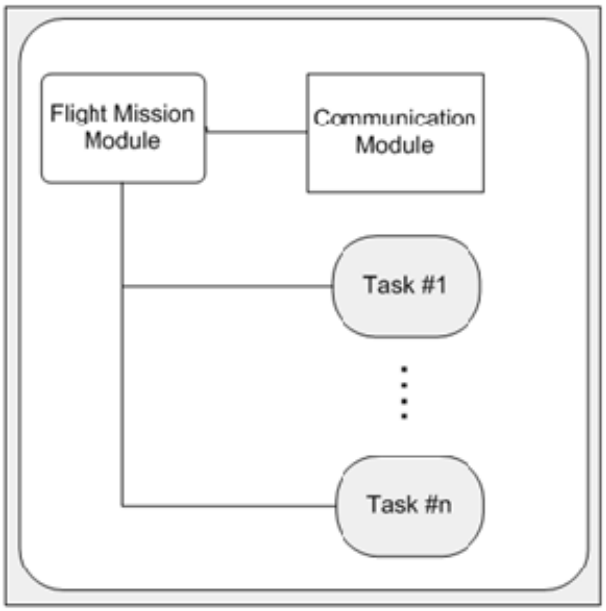

Figure 3.10 : Mission Coordination Computer Program Architecture

3.3.3 Virtual Manned Vehicle Simulation

Virtual manned vehicle simulation has the same architecture as in the VUV simulation, except that pilot controls the dynamic model that runs in this simulation manually. Therefore, there are several components that are used to take the reference inputs for the vehicle and inform the pilot about mission, visualization. In addition to the previously described components in the VUV simulation, there are xx new components: C2I module, xPC computer/Joystick, Panel A/D IO, Display computer, 3D Multi-monitor visualization system

Figure 3.11 shows the components of the VMV simulation below.

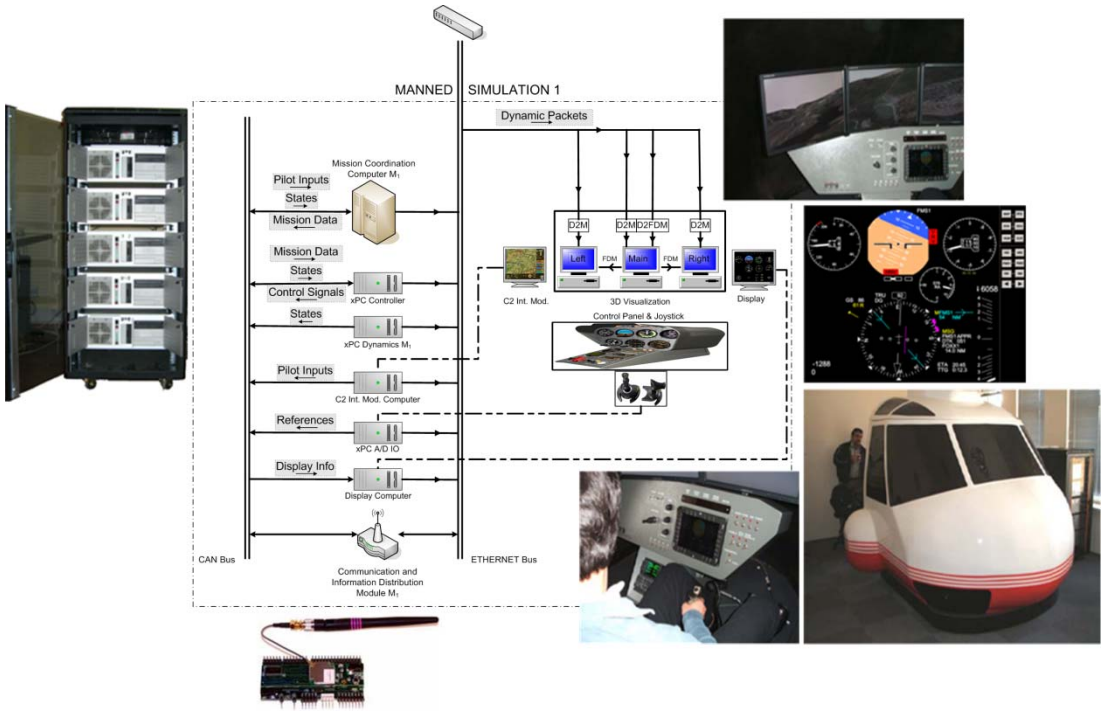

Figure 3.11 : Components of the manned simulation

Mission Coordination Computer: This computer is the same as the computer in the VUV simulation.

Command and Control Interface Module: The C2I module is a GUI based program that interacts with pilot. This interface takes states of the dynamic model of each vehicle and other commands from different components of network simulator as inputs, and then serves this information on a 2D map to the pilot as basic as possible. This interface mainly has two parts: a 2D map which shows other vehicles, waypoints, trails, and command panel that contains several commands like assigning a vehicle for a specific task.

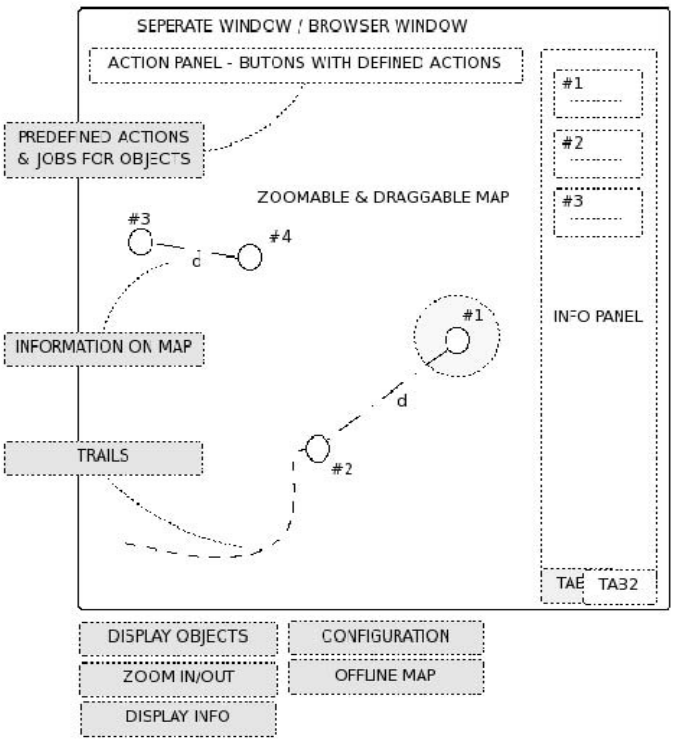

Figure 3.12 : Command and Control Interface GUI

xPC Computer/Controller : This computer is the same as the computer in the VUV simulation.

xPC Computer/Dynamics : This computer is the same as the computer in the VUV simulation.

xPC Computer/Joystick, Control Panel A/D IO: This computer has the same property with the xPC computer in the unmanned simulator except that the dynamic model that runs on this computer is controlled manually by pilot. Therefore, the reference inputs for the vehicle are inputted by using joysticks. The flight panel and its connection is illustrated in the following figures.

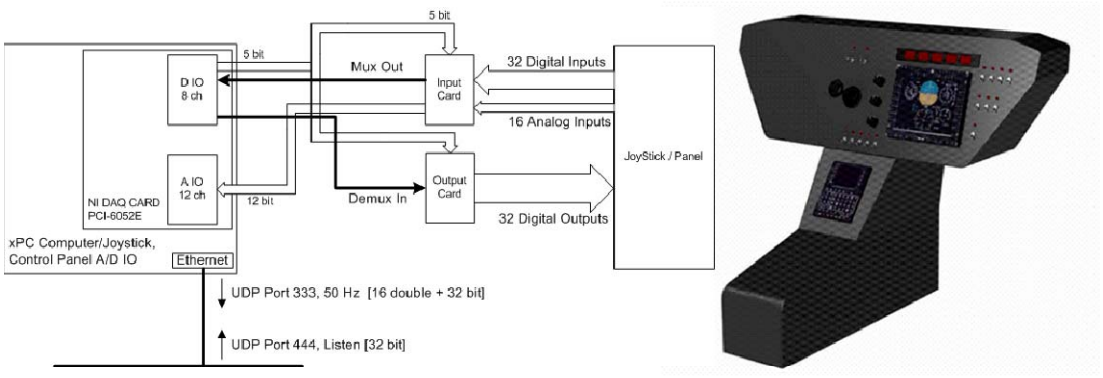

Figure 3.13 : Connection of the joystick and panel to the simulator

Display Computer: Display computer is a graphical user interface which displays the operator the selected air vehicle's flight related information. This information is displayed in 5 separate screens. These screens are MAP, Primary Flight Display (PFD), Synthetic Vision System (SVS), Engine Display Unit (EDU), and Health and Usage Monitoring System (HUMS). The operator can navigate though screens by an adaptive menu, which changes its navigation button positions and functions.

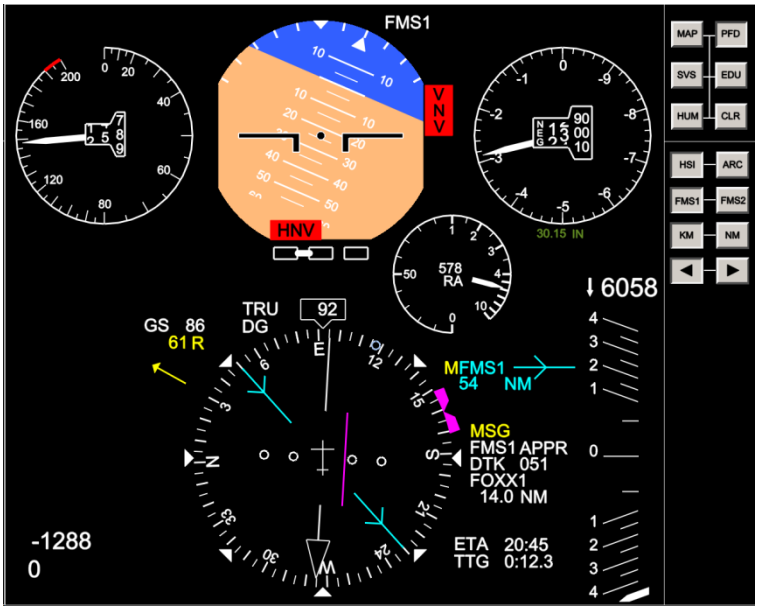

Figure 3.14 : Primary Flight Display

3D Multi-monitor Visualization: Multiple Display System enables 3D cockpit simulation in FlightGear environment. Multiple monitor configurations require 3 separate personal computers connected with each other through a network. Multiple Display synchronization property of FlightGear provides Flight Dynamics exchanging between two or more FlightGear programs running in different computers. Computers used for multiple display system are equipped with high capacity graphics cards, and these cards are connected to wide screen LCD monitors. Each of these monitors are configured and located to certain positions for establishing realistic cockpit simulation.

FDM Packets: In cockpit simulation mode, two computers run their simulation with the FDM packets that they receive from the other one which is accepted to be the central (or main) computer. Specifically for simulator, central computer running a FlightGear captures the flight dynamics of the current aircraft, and sends them to the Left and Right computer over Visualization Layer. In order to give some specific

information about multiple display synchronization, command line arguments for three FlightGear environments running in different computers are given below:

```
CENTRAL(MAIN) Computer :
--callsign=main   --native-fdm=socket,out,60,LEFT-IP,5500,udp
--aircraft=bo105 --native-fdm=socket,out,60,RIGHT-IP,5500,udp

LEFT Computer :
--callsign=left --native-fdm=socket, in,60,,5500,udp
--aircraft=bo105 --fdm=external

RIGHT Computer :
--callsign=left --native-fdm=socket, in,60,,5500,udp
--aircraft=bo105 --fdm=external
```

- "-fdm=external" command line argument is used to tell FlightGear that flight dynamics of "bo105" aircraft will be acquired from external computers.
- "60" is the predefined packet exchange frequency(Hz).
- Communication is done with User Datagram Protocol(UDP) using 5500 as the application number (port).

CID Module: This computer is the same as the computer in the VUV simulation.

3.3.4 Real Unmanned Air and Ground Vehicle Simulation

The architecture of the simulator is very extensible and module and it is possible to integrate real vehicles to the simulator without any extra modification. This is one of the unique features of the simulator which provides a better testing environment for HIL systems and planning algorithms. Figure 3.15 shows how the real vehicles can be connected to the network.

Mission Coordination Computer: Instead of a desktop PC, an embedded PC104 computer is used to run the planning algorithms.

MPC555/Controller: Instead of an xPC computer, an embedded MPC555 computer is used to run the real time controller algorithms. Since this embedded processor can also be programmed by Matlab/Simulink, there is almost no need extra effort to modify the controller algorithm used in xPC computer.

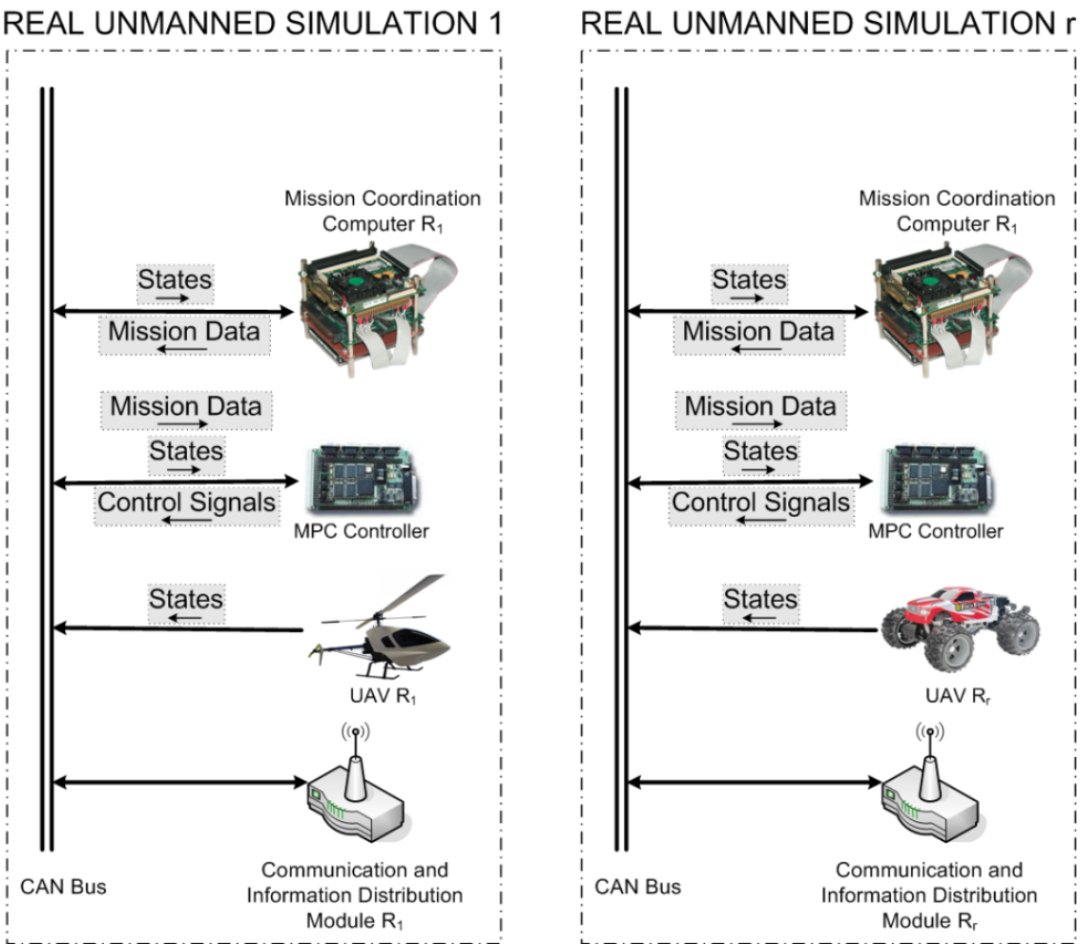

Figure 3.15 : Components of Real Unmanned Simulation

CID Module: This computer is the same as the computer in the VUV simulation.

Real UAV/UGV and Hardware in the Loop Simulator: Real UAVs/UGVs are fully equipped with sensors, embedded processors and the architecture of microavinoics hardwares is illustrated in Figure 3.16. As it can be seen easily, there is a main CAN bus and all components controller; sensors communicate each other via this bus. Since each sensor has different type communication interface such serial, IC2, a simple circuit, named as SmartCan, is designed to connect these sensors to the CAN Bus. SmartCAN makes packet conversion between different types of communication interfaces and CAN interface. A human pilot via Switchboard can operate these vehicles remotely. The human pilot can take the control of the vehicle by using this Switchboard in a dangerous manner [11].

In addition, this platform can be used as hardware-in-the-loop simulator and these design concepts enable us to test the performance of real-time control algorithms and hardwares not only outside, but also in the laboratory.

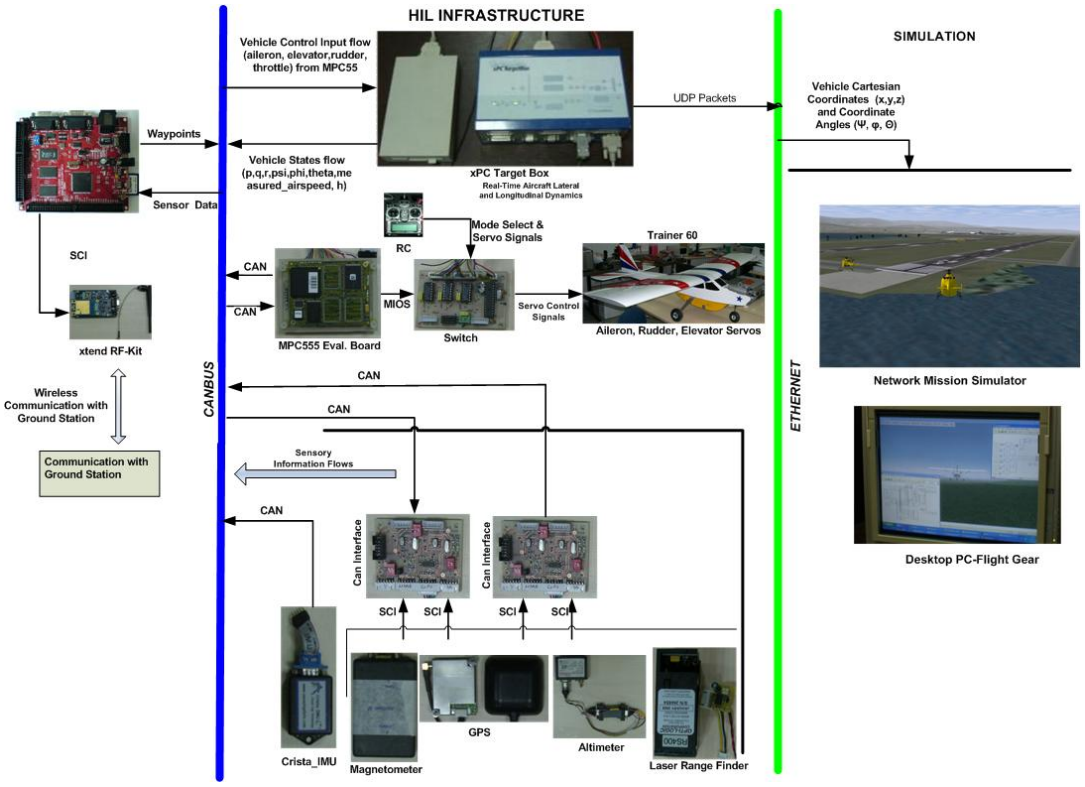

Figure 3.16 : Hardware in the Loop Simulator Design Structure

3.3.5 Ground Station

Ground station computer is used for tracking the mission, creating new mission, and sending special commands to any vehicle when mission is performed. This computer is also connected to the network via CID module like other vehicles.

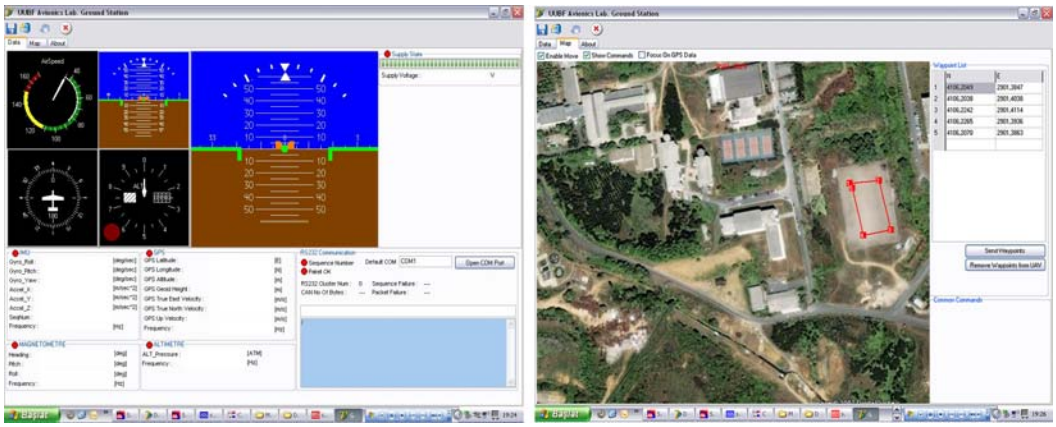

Figure 3.17 : Ground Station GUI

3.4 Communication and Information Distribution Module

The software of CID module is written in C++ and boost thread library is used for thread management [12]. General architecture of the software is developed as

modular as possible by using a highly object oriented design methodology. This modularity allows developers to design new classes and integrate them to the software rapidly [13]. Figure 3.18 illustrates an overview of the class hierarchy used in the software.

3.4.1 Software Architecture

Control & Forward Plane Level Classes

CommunicationModule Class: This class is used for configuring of the module. What it does are reading configuration data from a file and creating a controller object. Then it updates connection tables of the controller object according this configuration data and starts the controller object.

Controller Class: This class parses the control messages and performs the related operations indicated in the message. It has a list of control communication interfaces and polls them for new incoming messages in its update thread. If there is a new control message, first, it is parsed in order to understand meaning of the message, and then specific operations are performed. For instance, if there is a data request control message, first it checks whether the related data communication interface and forwarder objects are created or not. If objects are not created yet, update thread creates them and then it updates the routing table with new destination address information.

Forwarder Class: This class is responsible for distribution of data gathered from a communication interface to the destination addresses predefined in routing table. For each communication interface, one forwarder object is created and assigned to that interface by controller object. Based on some special data requests, this class has two kinds of thread which runs continuously or periodically. These threads take a data packet from incoming buffer of the related communication interface. Then, they check the routing table to obtain destination addresses for the related data packet. Finally, they send the data to the destination address by related communication interface periodically, or continuously.

- **Forwarder::Continuous Thread**

  This thread is created and starts when Start() method of the Forwarder class is called. Forwarder class has a routing table and address of a group CommunicationInterface objects. One of these communication interfaces is just

used for gathering incoming data packets, while the others are used for distribution of information. Thread performs its entire task in a loop, and it, firstly, gets an incoming data packet from incoming CommunicationInterface objects and parses it into its component. Then it sends data packet to the destination CommunicationInterface object. The simple routing algorithm is given in Appendix A.1.

- **Forwarder::Periodic Thread**

A periodic thread is created and stated for each dataID whenever routing information is inserted into the related dataID place of periodic routing table. basePeriod and sleepTime variables are used to be able to send a certain data (dataID) to the different destination addresses at different frequencies and these variable is updated whenever a routing information is inserted into or erased from the periodic routing table for considered dataID. Thread performs its entire task in a loop, and it, firstly, gets the latest incoming data from the buffer that is updated by continuous thread. Then it sends the data to the all destination address, if their period is expired. Algorithm for periodic operations is given in Appendix A.2.

UML diagram of the CommunicationModule, Controller and Forwarder classes is shown in the Figure 3.18.

Low Level Interface Classes

CommunicationInterface : Class This is an abstract class for implementation of different communication interfaces like CAN, ethernet, serial in order to make low level communication operations. Each communication interface class is derived from this abstract class and has two message buffers, namely incoming and outgoing message buffers. These buffers are used by calling Send() and Receive() methods of the interface class, then the update thread of the class processes the exact low level specific operations like send and receive operations, continuously. UML diagram of the CommunicationInterface class is shown in Figure 3.19.

Factory Pattern: An abstract factory pattern is used for creation of the CommunicationInterface objects.

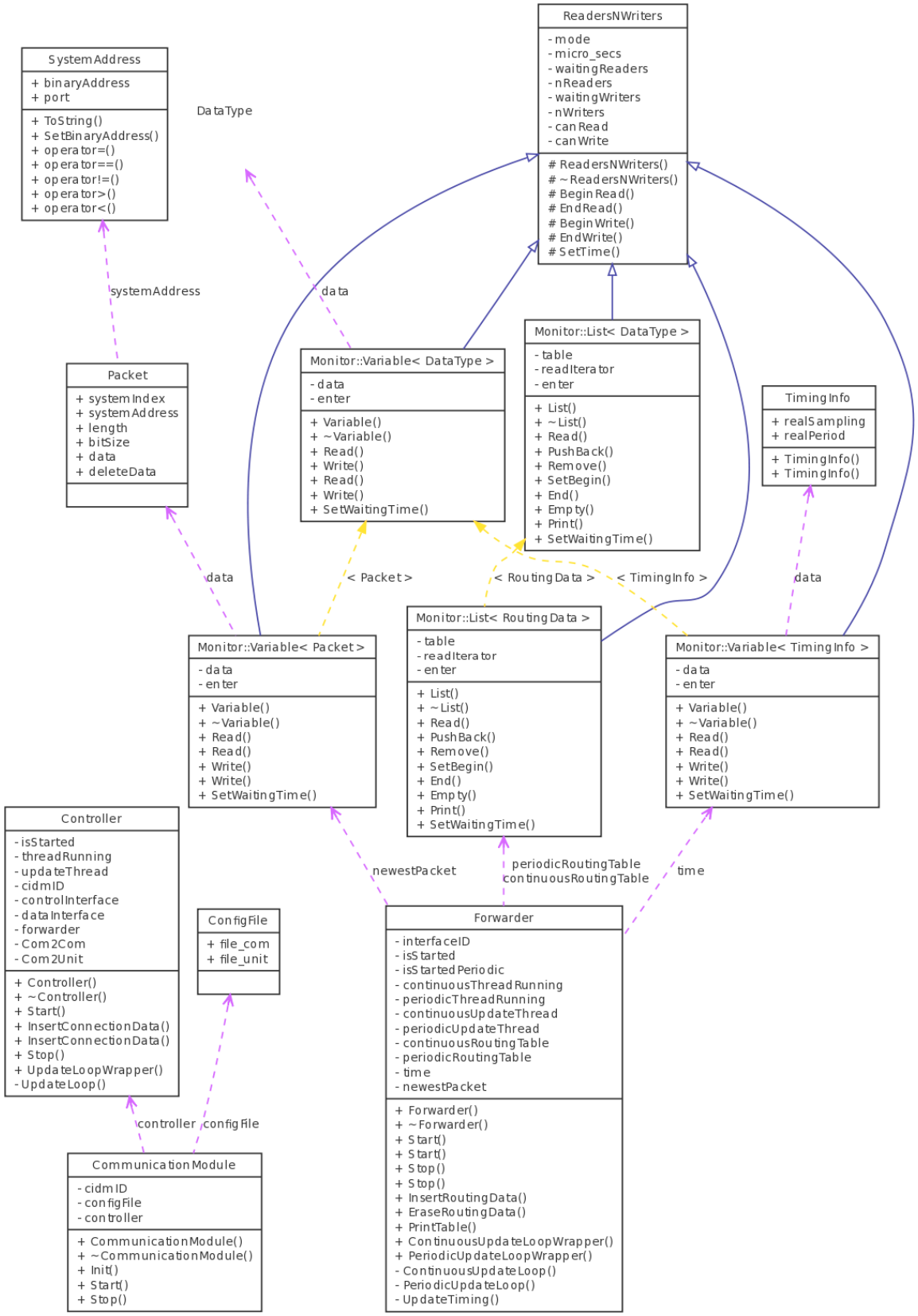

Figure 3.18 : UML diagram of the CommunicationModule, Controller, Forwarder classes

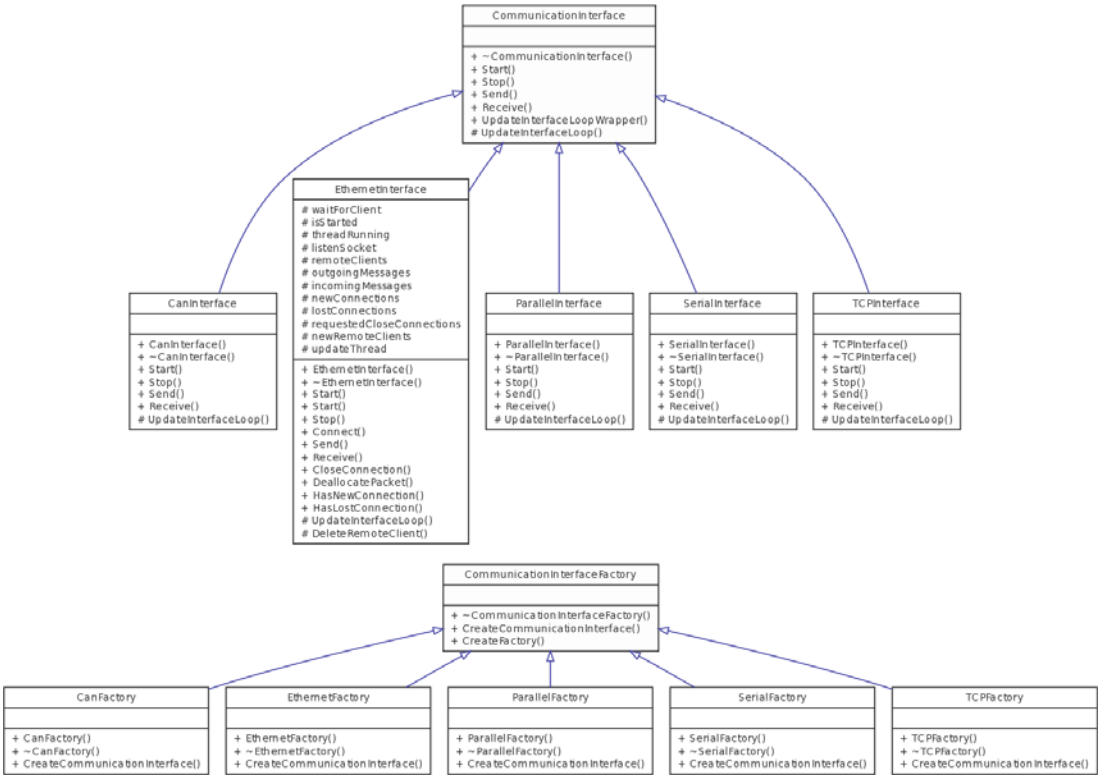

Figure 3.19 : UML diagram of the CommunicationInterface classes and factories

3.4.2 Classic Problems in Multi Thread Management

Since there are many threads need to be managed during execution of the program, classic thread synchronization problems exist such as producers & consumers, readers & writers problems. Monitor concept with Mesa semantic is used in order to solve these classic problems. Two base classes, namely, ProducersNConsumers and ReadersNWriters, are defined and they handle primitive lock/unlock operations.

Producers and Consumer Problem: This problem occurs between update threads of Forwarder and Controller objects and update threads of CommunicationInterface object. It can be easily seen in Figure 3.20 that these threads communicate with each other through incoming and outgoing messages buffers. Since there are too many threads perform on these buffers, it is essential to synchronize threads before inserting a message to buffer or removing a message from it. The class hierarchy of the monitor classes is shown in Figure 3.21.

Reader and Writers Problem: This problem occurs between update thread of Controller object and periodic and continuous update threads of Forwarder objects since they read and write on routing tables concurrently. In addition, there is another readers and writers problem between periodic and continuous update threads of each

forwarder object, since they share a message buffer which contains up-to-date packets. The class hierarchy of the monitor classes is shown in Figure 3.22.

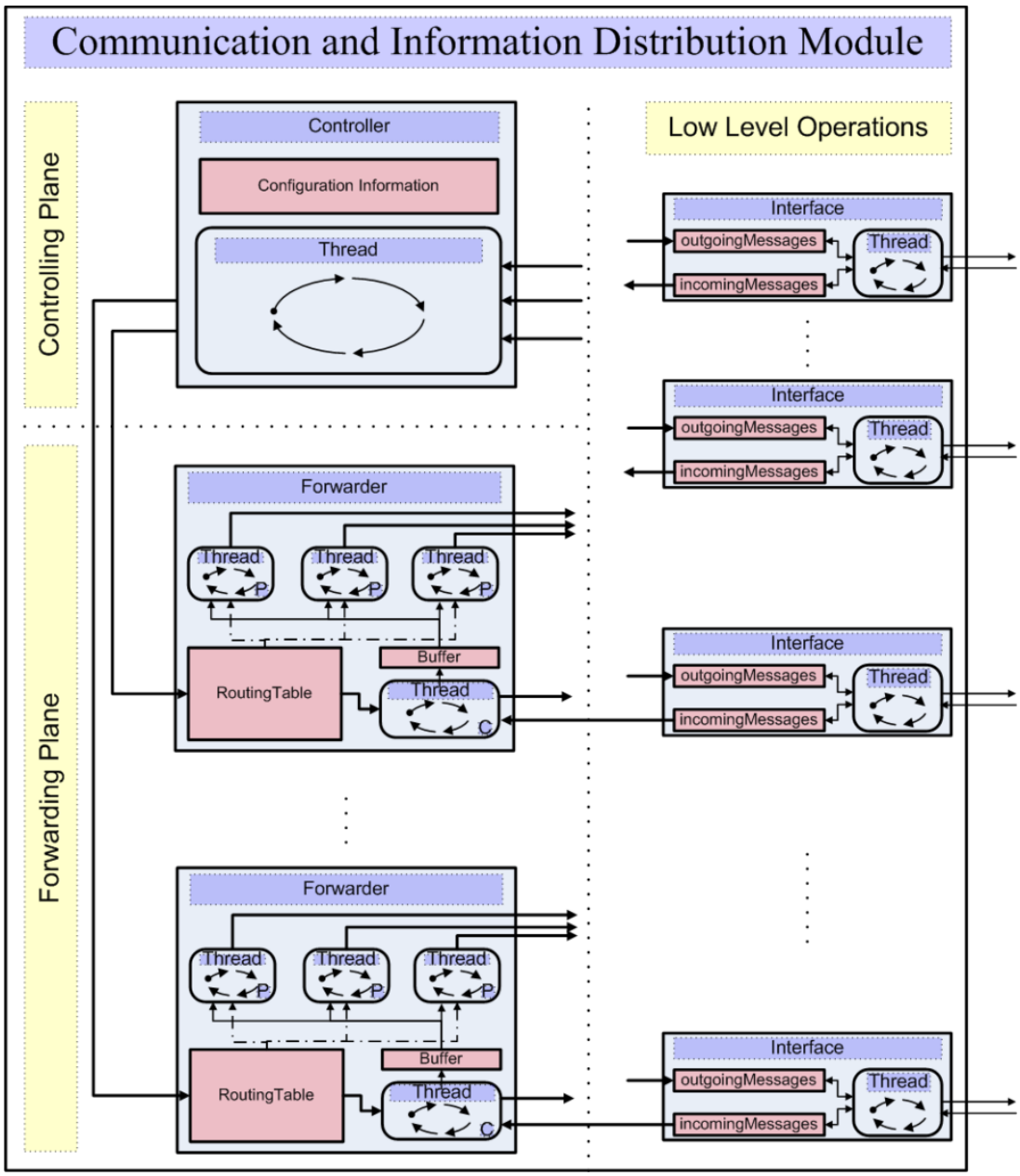

Figure 3.20 : Thread interactions and shared data

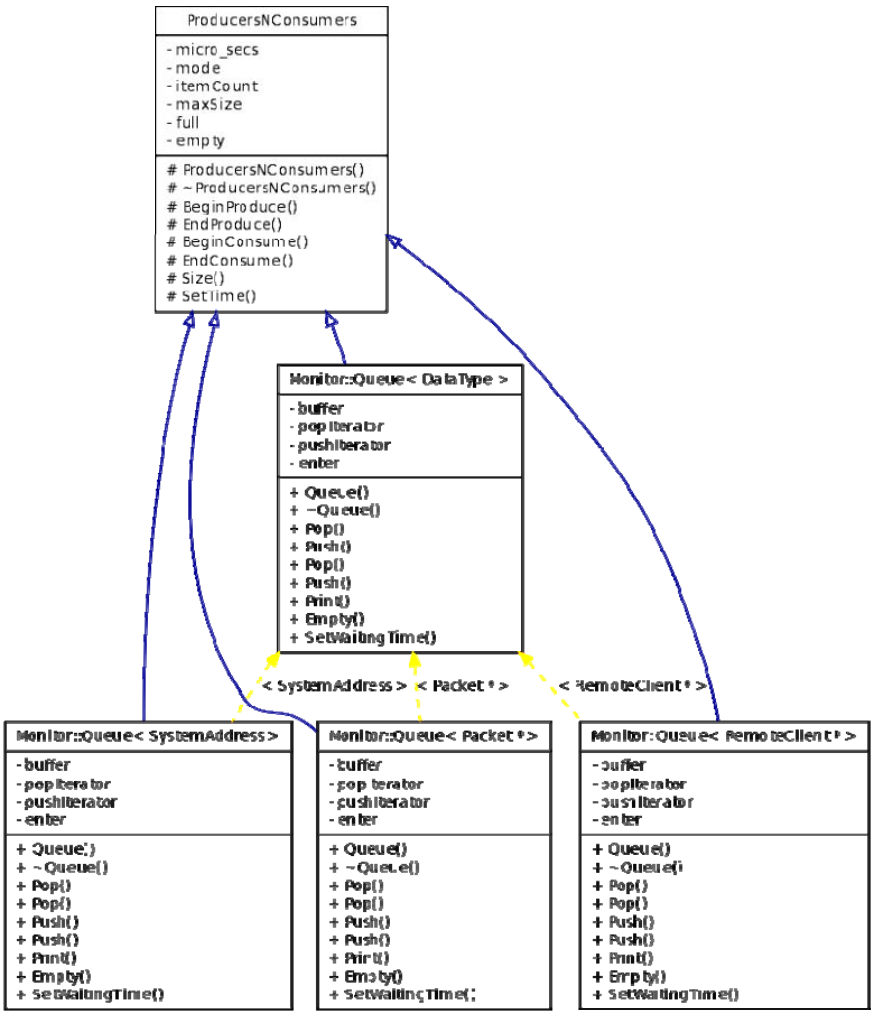

Figure 3.21 : UML Diagram of ProducersNConsumers class and other monitors

3.5 Testing

Besides unit test of all classes, overall module is tested under the several cases. The conceptual CUMV network illustrated in Figure 3.23 is simply built in order to test the transmitting data between not only similar units, but also dissimilar units.

Test cases for transmitting data

- (Ground Station --- CID Module) --- (CID Module --- MCC)
- (MCC --- CID Module) --- (CID Module --- MCC)

Information flows in the specified test case are shown in Figure 3.23.

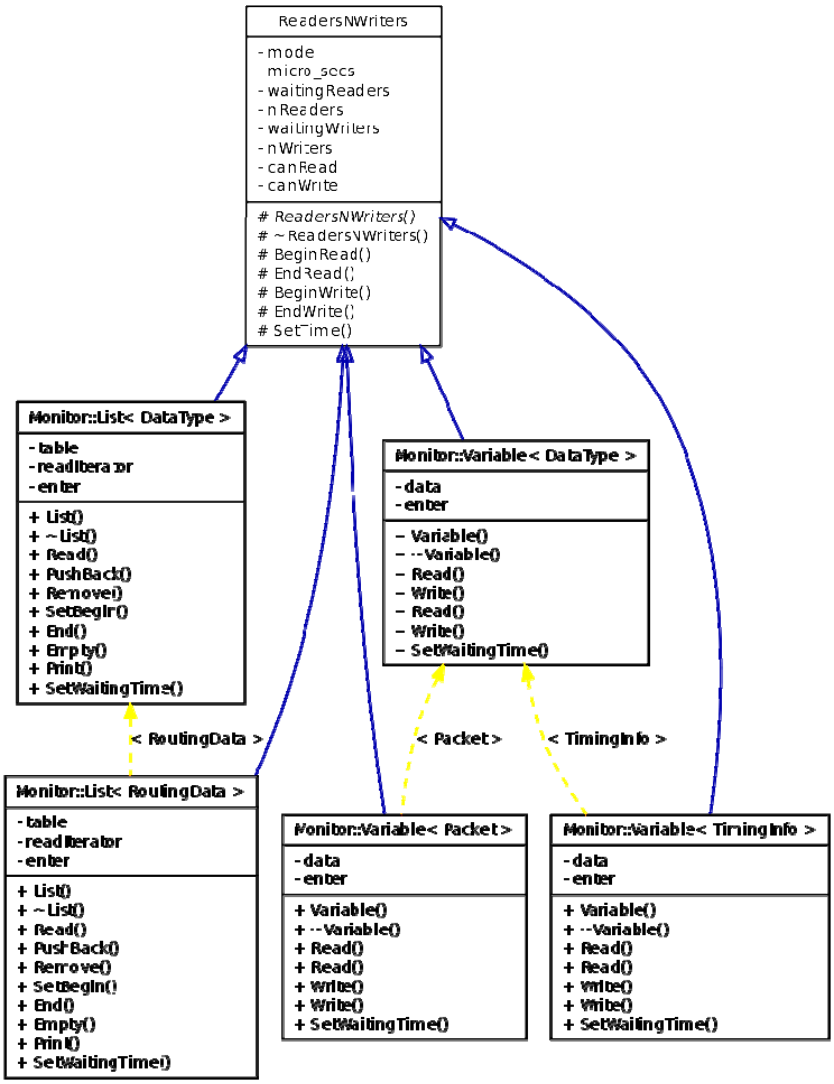

Figure 3.22 : UML Diagram of ReadersNWriters class and other monitor classes

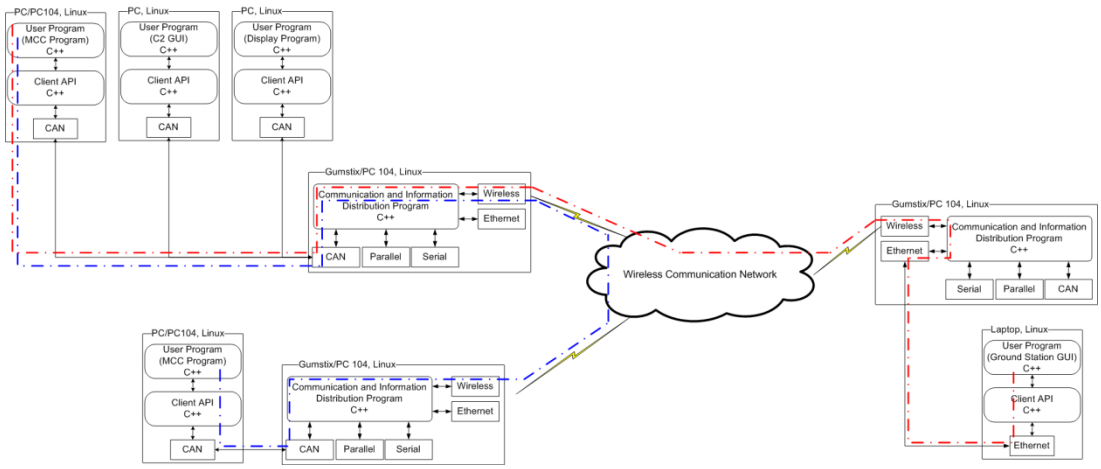

Figure 3.23 : Test cases for transmitting data

4. IMPLEMENTATION OF DECISION SUPPORT ALGORITHMS

Growing involvement of UAVs in complex application areas (such as dynamically changing urban rescue operations), the types and the number of tasks easily outgrow one vehicle's (or a set of UAV operators' command and control) limited capabilities and thus require a fleet of UAVs to work in a collaborative fashion to achieve desired operational goals. Despite the current trend reducing the role of human in aerial systems, human operators are still needed for supervisory control and high-level decision making [14]. In this section, we provide an event driven decision support algorithm for temporal scheduling of tasks across UAV assets based on Solve & Robustify approach [15] [16]. Specifically, the algorithm is designed and tested to provide hard real-time decision support (i.e. on the order of seconds) to the operator for scenarios involving over hundred tasks across multiple assets.

A key part of such an event-driven process hinges on the coordination of the target/task assignments and distributions across UAV assets through supervisory commanding [17]. However, it is not always feasible to apply basic supervisory command and control for a large number of unmanned vehicles performing complex missions with strict time constraint. Therefore, a decision support or autonomous decision-making system can be designed for the commander that decreases the workload. Figure 4.1 illustrates the overall process of such a decision making support system. Such a decision support system requires integration of two important operations: planning -the problem of determining which activities to perform, and scheduling - the problem of temporal allocation of resources to these activities [18].

The steps of decision making can be tracked as follows: planning, scheduling and low-level mission specific task planning. The planning process is triggered by an external monitored event such as bombing, threat detection or a task request from mission control center. Following the taxonomy as presented in [1], the planning step proceeds by selecting suitable actions (UAV missions) from a predefined action sets by an expert system. Then, resource allocation is done for these action sets regarding operation constraints (time, environment) and a group of target is selected regarding

to the selected actions and resources. The second process is scheduling of these action sets under resource constraints by satisfying specified time windows constraints. In this thesis, we assume that the set of activities requiring resources are specified in advance and focus on temporal allocation of a set activities regarding resource and strict time constraints as fast as possible. Specifically, to handle executional uncertainty in the dynamic mission environment, Solve & Robustify approach is used as a base algorithm. The algorithm used in the Solve step, Earliest Start Time Algorithm (ESTA) [19], is modified with temporal space partitioning to provide real-time solutions to the operator. Benchmark problem comparisons with the classical ESTA formulation for two hundred tasks indicates that the proposed temporal space partitioning approach improves the computation time forty-fold while only incurring five percent increase in the total completion of the tasks. After successful temporal allocation of the actions, the low-level task planning problem is solved by the algorithm given in [20].

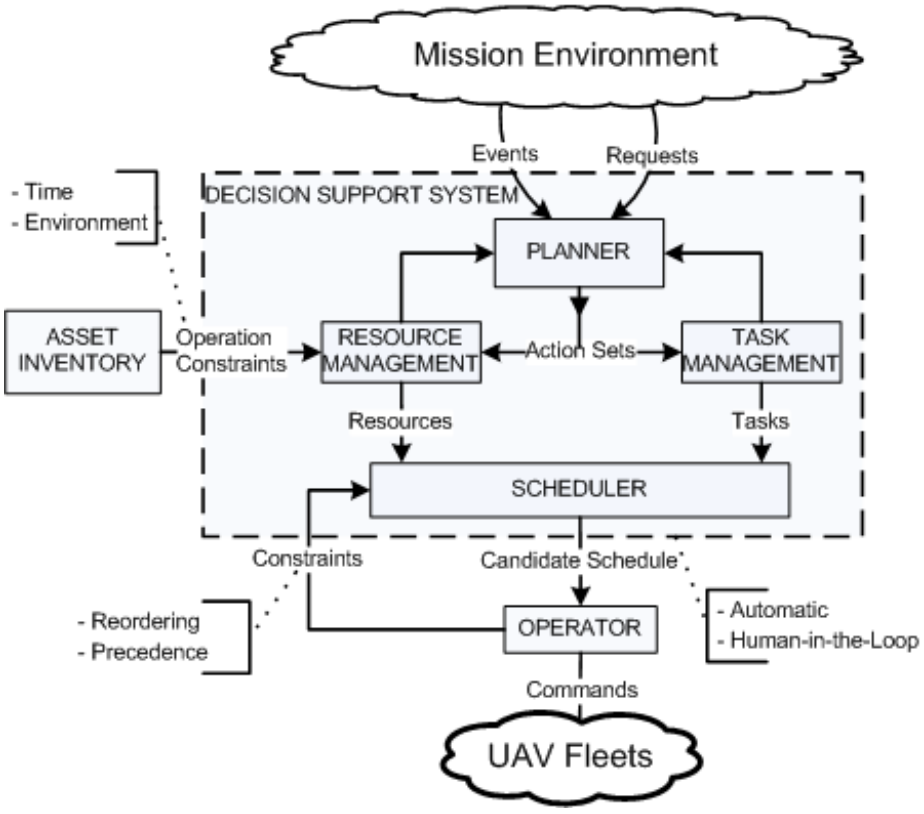

Figure 4.1 : Event-Driven Decision Support Architecture

The organization of this section is as follows:

First, the scheduling step of the decision support system is formulated as a classical Resource Constraint Project Scheduling with Maximum Lag (RCPSP/max) problem.

A brief literature survey about different approaches for solving this classic problem is given. Specifically, Solve & Robustify approach is introduced for robust and flexible schedule generation. Following a brief computational analysis of Solve & Robustify, the temporal space partitioning is proposed for decreasing the computationally expensive solve step. Then, the basics and the fundamental operations of the ESTA algorithm are reminded. Specifically, construction of an infinite capacity solution and leveling of resource demand by posting precedence is used in the proposed greedy scheduling algorithm ($ESTA_P$). After that, the partitioning heuristics used in the algorithm and the overall greedy algorithm is described gradually. Finally, in the last section, experimental evaluations and results are given that demonstrate the efficiency of the proposed algorithm across a range of benchmark problems of increasing activity size.

4.1 The Scheduling Problem RCPSP/MAX and Solution Approaches

Figure 4.2 illustrates an overview of distributed of command and control (C2) problem in which missions are accomplished by human operators and UAV fleets. All entities, both humans and vehicles, can be considered as members of a cooperative team working towards accomplishing a common goal. From the point of the human operators' view, the C2 problem involves scheduling of several missions consistently regarding to the number of UAVs and temporal constraints between the missions. This problem can be defined as a scheduling problem by using the following definitions:

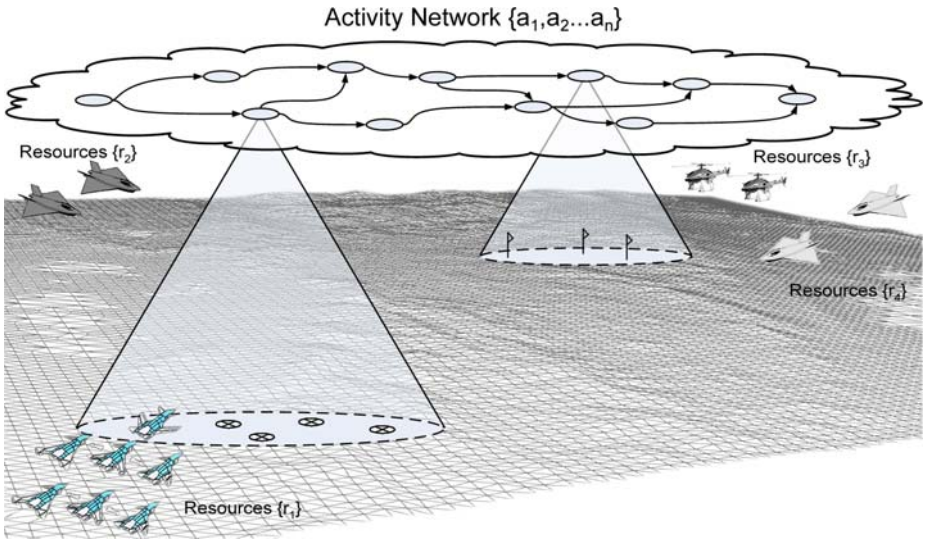

Figure 4.2 : Mission Environment

- Each mission (i. e. reconnaissance, intelligence, surveillance) corresponds to an activity which has to be scheduled and has an estimated duration
- UAV fleets correspond to a set of renewable resources and each UAV with different capabilities (i.e. combat, imaging) define a new type of resources
- Each mission requires a number of UAVs for all its duration
- Structural dependencies between missions, physical and logistic constraints define a set of temporal constraints. For example, a target must be destroyed (attack) within some periods of time immediately after its designation (reconnaissance), the total completion time of set of missions assigned to a UAV can not exceed its endurance and it must return to base for maintenance before a certain amount of time from endurance.
- Each low-level mission specific task corresponds to a job. For example, designation of a set of targets defines a mission which consists of several tasks of visiting each target. These tasks correspond to a set of jobs that must be processed in such order to complete the mission.

This scheduling problem can be stated as finding a temporal allocation and synchronization of a set of renewable resources $R = \{r_1 \dots r_m\}$ for given a set of activities (missions) $V = \{a_1 \dots a_n\}$ over time regarding a set of temporal constraints. In this work, we use the Resource-Constrained Project Scheduling Problem with Minimum and Maximum time lags (RCPSP/max) as a reference [21] and each activity must be performed regarding the following constraints:

- each activity a_j has a duration dur_{a_j}, a start time s_{a_j} and an end time e_{a_j} such that $e_{a_j} = s_{a_j} + dur_{a_j}$;
- each activity a_i requires the use of req_{ik} units of the resource r_k for all of dur_{a_i}
- a set of temporal constraints c_{ij} each defined for a pair of activities (a_i, a_j) and of the form of $c_{ij}^{min} \leq s_{a_j} - s_{a_i} \leq c_{ij}^{max}$
- each resource r_k has an integer capacity $max_k \geq 1$;

The objective is to determine the starting times of all activities in V, $S = (s_1, s_2, \dots, s_n)$, which are temporally consistent and satisfy resource capacity constraints.

There are different models and approaches for solving RCPSP/max in Operations Research community. One the most used approach is solving the problem in branch and bound schema. An early analysis focusing on mathematical properties of the problem was given in [21] and a branch and bound approach was proposed which is based on extending the set precedence relations in order to resolve conflicts in the problem. In that work, the solution space was modeled as a network of temporal constraints and the concept of *forbidden sets* and *reduced forbidden sets* to define resource conflicts and resource conflicts with a minimal number of activities respectively. Then a systematic Branch and Bound (B&B) was formalized as posting precedence relations in the problem to remove all reduced forbidden sets in an initial, time feasible solution step by step. Patterson et al proposed another B&B algorithm based on precedence tree. In [22], [23] and improved by additional dominance rules in Sprecher [24].

Another approach for solving scheduling problem is formulating RCPSP/max as constraint satisfaction problem and there are two main models in the CSP scheduling literature: start time assignment [25], [26] and precedence constraint posting model [27], [28]. In the first model, time points that indicate the start times of activities are defined as decision variables of the problem and aim of the CSP search is finding a consistent (both time and resource feasible) assignment of start time values. In the second case, various ordering decisions between sets of activities that are competing for the same resources are considered as decision variables rather than start times of activities and the aim of the CSP search is posting a consistent set of precedence constraints that removes all the resource peaks in the resource profiles of the resources. Since, in this approach, there is no any specific start time assignment for each activity during the search process or in the final solution, it is less commitment than the first model. As a result, in the uncertain dynamic environment, this model has advantages as having capability of generating more robust and flexible schedules than the first model.

Due to requirement of robust and flexible schedules to handle executional uncertainty in the dynamic mission environment, one of the successful CSP based algorithm Solve & Robustify approach given in [15], [16] is used as base algorithm and the proposed temporal space partitioning method is implemented within the Solve & Robustify approach.

The Temporal Space Partitioning

During the implementation of the Solve & Robustify approach, it is observed that the first step, which is computation of a fixed-time solution increasingly, dominates the overall solution time. The most of computation done in this step is contributed by leveling resource demand by posting precedence. The leveling resource demand operation basically consists of two steps, namely, determination of which precedence to post and finally propagating the precedence constraint along the temporal network.

First, all contention peaks[1] are collected for a given earliest start solution. Then, Minimal Critical Sets[2] (MCSs) are computed for each peak. Since the complete analysis of MCSs has exponential computation in nature, a *sampling strategy* of polynomial complexity (e.g., $O(n)$, $O(n^2)$) given in [29] is used to collect some subset of MCSs for each peak by abstaining from combinatorial explosion of the search. Unfortunately, as the size of the problem increases regarding to the both number of resource types and number of temporal variables in the network, the number of peaks for a given earliest start solution increases too fast in most cases. This tends to increasingly spend too much time for collecting of MCSs even if using sampling strategies of polynomial complexity.

Second, after selecting the MCS to be resolved, this MCS is solved by posting a simple precedence constraint between pair of competing activities. In order to maintain the consistency of the problem's temporal information, that constraint is propagated along the temporal network by using all pairs shortest path algorithms, which are $O(n^2)$ in space and $O(n^3)$ in time according to the number of temporal variables in the network. Unfortunately, as problem size increases, computation of propagation of the constraints dramatically takes too much time.

[1] Contention peak is a set of activities which are competing for the same resources and their simultaneous execution exceeds the resource capacity

[2] A Minimal Critical Sets, is a contention peak such that any subset of activities belonged to MCS is not a contention peak

Thus, in order to solve the UAV mission inherent large-scale scheduling problem in real-time, we propose an approach that divides the larger problem into smaller subproblems by partitioning the temporal space and then iteratively solving the subproblems. To obtain balanced partitioning, a temporal space flattening method is developed. This method eliminates overlapping of conflicting activities through temporal sequencing in an expanded time space. To obtain a consistent integration of subproblem solutions, a binding method is introduced. This method propagates the maximum horizon constraint over the whole network to generate the main solution.

These methods are explained in the following section in detail.

4.2 A Greedy Algorithm Based on the Temporal Space Partitioning

Temporal space partitioning approach consists of three basic steps: (a) a partitioning algorithm is first used to divide the problem into subproblems by partitioning the temporal space; (b) the first subproblem is solved recursively by using the same approach; then the second problem is redefined with regard to the updated partial solution and finally the second problem is solved in the same way; (c) the solutions obtained from first and second subproblems are integrated to maintain consistency in temporal space.

The proposed approach is based on the Earliest Start Time Algorithm (ESTA), given first in [30] and further improved in [29]. The steps of ESTA are summarized as follows and these steps are separately used in the proposed algorithm.

4.2.1 Summary of ESTA Procedure

Construct an infinite capacity solution

In this step, the problem is formulated as an STP [31] temporal constraint network and a time feasible solution that ignores resource constraints –assuming infinite resource capacity- is computed by propagating constraints on the underlying temporal constraint network.

Level resource demand by posting precedence

Resource demand profile for each type of resources is computed for all time values and the time intervals that resource demands exceeds resource capacity are detected. Finally, these detected resource conflicts are resolved by iteratively posting simple precedence constraints between pairs of competing activities.

4.2.2 Separating Steps of ESTA

The steps of *ESTA* procedure are separately used in the proposed algorithm which is referred as $\boldsymbol{ESTA_P}$ where the suffix *P* indicates the partitioning. As it can be seen easily in Appendix A.3, before solving the problem recursively, an infinite capacity solution is computed by the first step taken from *ESTA* procedure. Then, the resource demand of this partial solution is leveled until there is no any resource conflict in its earliest start solution by calling $ESTA_{PR}$ procedure.

In the *ESTA* procedure, the resource demand of earliest start solution is leveled by posting simple precedence constraints until eliminating all resource conflicts. However, this is required only when the size of the subproblem is small than predefined minimum size value in the proposed approach. Otherwise, as it is shown in Appendix A.4, the leveling step is repeated for a given number of times as required in special circumstances.

4.2.3 Solving the Problem by the Temporal Space Partitioning

Posting precedence constraints and propagation of them are fundamental operations in ESTA and it is seen that propagation of constraints over the temporal network sometimes does not cause an update in the temporal information of some activities during the implementation of ESTA. Actually, the effect of propagation of constraints over temporal networks is closely correlated with how deeply the activities of temporal networks are interconnected. The distinction of sensitive and less-sensitive parts of temporal networks to constraint propagation are more evident in the temporal networks which are sufficiently flexible and have enough temporal slacks among their activities. The temporal space partitioning approach is based on exploiting such properties of temporal networks by dividing them into subnetworks. Appendix A.5 shows the $ESTA_{PR}$ procedure where P and R indicate partitioning and recursive, respectively in more detail. It takes two parameters as input: (1) a partial solution of the problem P S^P, and (2) maximal horizon of the problem P h_{max}.

At the first step, the size of the problem with regard to the number of the activities is compared with the minimum size value. In the case that the size of the problem P is less than minimum size, then all the resource conflicts detected in the partial solution S^P are eliminated by the LevelResourceDemand procedure at Step 2. Otherwise,

since the size of the problem is bigger than the minimum size, it is divided into subproblems by partitioning the temporal space of problem *P*.

The temporal space of problem *P* is divided into 2 subregions by cutting it from the time point t_c, where suffix *c* indicates critical within the loop at Step 6. An example of partitioning of the temporal space by $t_c = 9$ is illustrated in Figure 4.3. First, the critical time point t_c is computed by *CreatePartition* procedure at Step 7 and then the first subproblem is created by collecting activities from problem *P* such that their earliest start time value is less than t_c. If there is any activity in the first subproblem and the size of the first subproblem is less than the size of the problem *P*, then the inner loop is broken. Otherwise, this means that configuration of the current temporal space of problem *P* is too shrunk and it needs to be flattened. Flattening of the temporal space increase the divisibility and it is flattened by posting one precedence constraint by *LevelResourceDemand* procedure at Step 12. The inner loop repeats until dividing the problem *P* into smaller subproblems.

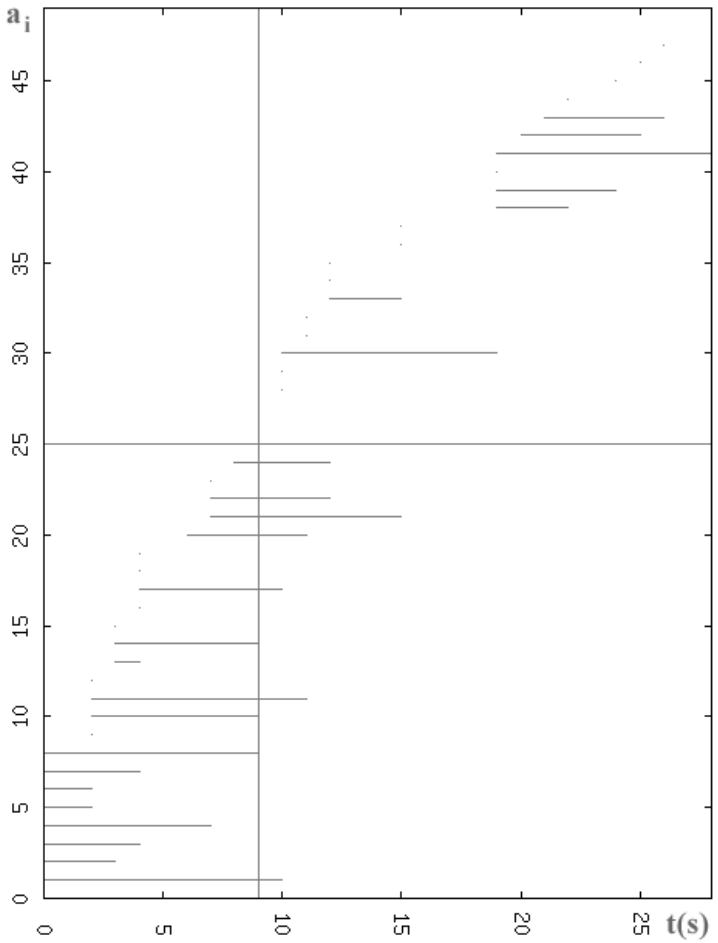

Figure 4.3 : The temporal space partitioning

After successfully dividing the problem *P*, the current state of the partial solution S^P is saved by assigning it to S_0^P at Step 14. This backup operation is necessary for the case that integrated solutions of subproblems leads to a time infeasible solution for problem *P*. While the first subproblem P_1 includes start time of the problem *P*, it has no any dummy finish time point. Therefore, a dummy finish time point t_{de} is included in subproblem P_1 at Step 15 and the horizon constraint between t_s and t_{de} is posted in the next step. Finally, the partial solution S_1^P is solved by calling the same procedure $ESTA_{PR}$, recursively. If there is a failure in the computed solution S_1^P, then the $ESTA_{PR}$ procedure returns FAILURE value. Otherwise, the $ESTA_{PR}$ begins to create second subproblem by collecting activities from problem *P* such that their earliest end time value is greater than t_c. Unlike to the case in the first subproblem, a dummy start time point t_{ds} is included in the subproblem P_2 at Step 22, since the second subproblem P_2 includes the end time of the problem *P* and then the horizon constraint between t_{ds} and t_e is posted. As in the case of the first subproblem, the partial solution S_2^P is solved by calling the same procedure $ESTA_{PR}$, recursively. If there is no any failure in the computer solution S_2^P, then these two computed solutions is integrated by simply propagating the updated temporal information along the temporal network of problem *P* via all pairs shortest path algorithm at Step 28. If a time feasible solution for problem *P* is gained, then $ESTA_{PR}$ procedure returns this solution. Otherwise, the S^P is assigned to its previously saved state S_0^P at step 34, then temporal space of problem *P* is flattened by posting a simple precedence constraint by *LevelResourceDemand* procedure at step 35. The outer loop is repeated infinitely until a time feasible solution exists or a failure is encountered in the solution of the subproblems.

4.2.4 Partitioning Heuristic

The partitioning heuristic is based on dividing the temporal space of the problem by a critical time point t_c such that the resulted subproblems are balanced with regard to the number of activities. The number of activities in the first subproblem is calculated by considering the earliest time values of start time variables of activities and the first subproblem consists of activities such that their earliest start time value is less than earliest time value of the t_c. However, the normalized percentage of the domains of the time values respect to the t_c is calculated for the second subproblem

instead of number of activities. The main reason is that it is not as certain as in the first subproblem to determine which activities will be collected to solve for second subproblem. The certain number of activities in the second subproblem only can be determined by checking the $est(e_{a_i}) > t_c$ for all activities in the problem after solving the first subproblem. So that the calculated number n_2 is just approximate value about the number of activities of second subproblem. After calculating n_1 and n_2 a cost value is calculated for t_c and for the simplicity the cost function is selected as $\frac{n_1^3+n_2^3}{n^3}$ inspired from the propagation complexity ($O(n^3)$) of precedence constraints over temporal network. Finally, CreatePartition procedure shown in Appendix A.6 returns the earliest time value of t_c with the minimum cost values.

4.3 Experimental Results and Evaulation

We have compared the performance of the $ESTA_P$ and *ESTA* procedure on the three sets of benchmark problems taken from the RCPSP/max problem repository [32], [33]. These sets are *UBO50*, *UBO100* and *UBO200* of 90 instances of problem of different size 50 x 5, 100 x 5 and 200 x 5 (number of activities x number of resources). The numbers of solvable instances are 73, 78, and 80 respectively. The rest of the problems have provably infinite lower bounds for makespan.

4.3.1 Experimental design

Both algorithms, $ESTA_P$ and *ESTA* are implemented in C++ using Eclipse IDE and the CPU times given in the table are measured on a PC with Intel Pentium(R) D CPU 3.40 Ghz processor under Fedora Core 9. The parameters of MCSs sampling strategy are set to $\delta \leftarrow 2$ and $s_f \leftarrow 100$ values. The maximal horizon h_{max} is set to 5000 in order to find a solution quickly by searching within a sufficiently large horizon.

4.3.2 Evaluation criteria

The following performance measures are calculated for comparative analysis of both algorithms on different problem sets:

- N_{feas}% the percentage of problems feasibly solved for each benchmark set
- t_{mks} average makespan of the solutions

- t_{cpu} average CPU-time in seconds spent to solve instances of the problem set
- N_{pc} the number of leveling precedence constraints posted to solve a problem
- $\Delta_{LB}\%$ the average of percentage relative deviation from known lower bound

First, both algorithms are not able to solve all the problem instances in the benchmark set due to subset of instances which their infeasibility is proven by branch and bound algorithm. However, they solve all the problems which have a finite lower bound found by several algorithms [33].

Second, the average makespan of the solutions found by the $ESTA_P$ algorithm is slightly larger than the ones found by *ESTA* algorithm and this observation results in slightly larger deviation from lower bound of the solutions. However, there is a major improvement in the average computation time, and the number of posted constraints in $ESTA_P$ over *ESTA* and the $ESTA_P$ algorithm dominates the *ESTA* algorithm across all problem sets for t_{cpu} and N_{pc} metrics as the size of the problem increases.

Table 4.1 : Performance of the algorithms (UBO50)

UBO50	t_{mks}	$\Delta_{LB}\%$	t_{cpu}	$N_{feas}\%$	N_{pc}
$ESTA_{P11}$	217.671	28.757	0.360	77.778	54.471
$ESTA_{P15}$	217.099	28.247	0.383	78.889	55.141
$ESTA_{P16}$	217.306	27.531	0.400	80.000	56.694
$ESTA_{P17}$	**218.973**	**27.459**	**0.373**	**81.111**	**56.904**
$ESTA_{P20}$	218.466	27.462	0.452	81.111	60.480
$ESTA_{P21}$	218.699	27.595	0.462	81.111	59.384
ESTA	**213.603**	**24.455**	**4.004**	**81.111**	**74.890**

The $ESTA_P$ is tested on the benchmark set UBO50 for different values of the minimum partition size and $ESTA_P$ improves the computation time ten-fold by decreasing it from 4.004 to 0.373 seconds. During the testing of the proposed algorithm, it is observed from Table 4.1 the performance of the algorithm is closely correlated the minimum partition size parameter. The performance of the $ESTA_P$ becomes identical with the performance of ESTA as the value of this parameter increases. However, if the value of the parameter decreases until a certain value, then the algorithm has a better performance in the sense of the computation time and

number of the posting constraints. However, if the parameter sets to too small values, then the $ESTA_P$ can not solve some instances of problems because of the inconsistency of time constraints, while the ESTA can.

Table 4.2 : Performance of the algorithms (UBO100)

UBO100	t_{mks}	$\Delta_{LB}\%$	t_{cpu}	$N_{feas}\%$	N_{pc}
ESTA $_{P12}$	**423.167**	**30.970**	**3.075**	**86.667**	**120.077**
ESTA $_{P15}$	419.705	29.833	3.121	86.667	123.538
ESTA $_{P20}$	418.436	29.697	3.166	86.667	128.910
ESTA $_{P25}$	419.141	30.100	3.345	86.667	132.974
ESTA	**407.286**	**25.645**	**79.214**	**86.667**	**195.753**

As seen in the Table 4.2 $ESTA_P$ has made great improvement in computation time by decreasing it from 79.214 to 3.075 seconds, while the resulted solutions have 5 percent larger deviation than the ones found by *ESTA*.

Table 4.3 : Performance of the algorithms (UBO200)

UBO200	t_{mks}	$\Delta_{LB}\%$	t_{cpu}	$N_{feas}\%$	N_{pc}
ESTA $_{P18}$	770.974	29.970	26.761	85.556	254.961
ESTA $_{P20}$	765.481	28.987	33.866	85.556	261.091
ESTA $_{P25}$	**763.474**	**28.603**	**31.782**	**86.667**	**275.897**
ESTA $_{P30}$	759.766	27.987	33.159	85.556	281.169
ESTA	**751.962**	**26.946**	**1462.83**	**86.667**	**461.436**

The $ESTA_P$ algorithm solves the problems of the benchmark set UBO200 in average 31.782, while the *ESTA* solves in 1462.83 seconds as shown in the Table 4.3. Thus, $ESTA_P$ algorithm reaches the highest rate of improvement of computation time, forty-fold, over the *ESTA* algorithm.

Finally, by using partition temporal space idea, we have developed fast algorithm which is based on *ESTA* algorithm and this algorithm can find a fixed solution very fast but with a slightly larger makespan.

4.4 Conclusions and Future Works

In this work, a large-scale UAV fleet-task scheduling algorithm is given for an event driven decision support system. Both the proposed algorithm $ESTA_P$ and one of the well-known algorithms *ESTA* are implemented in C++ and a comparative analysis is

given on sets of benchmark problems with increasing number of activities at the end of the paper. The results show that the proposed algorithm provides a significant improvement in computation time but founds solutions of slightly larger makespan. However, the enormous improvement on computation time provides us to solve the problem of large sized in real-time and this compensate the negative results of slightly larger makespan of the solutions.

During the testing of the proposed algorithm, it is observed that the performance of the algorithm is sensitive to the minimum partition size parameter. A good analysis about selecting value of the minimum partition size is left for future work. It may give a better performance by changing the value of the minimum partition size parameter dynamically for each subproblem. Other works that will be studied in the future are the followings: developing and modifying the proposed approach for low-level task planning (task/target assignment with timing constraints), development of the expert system to solve the planning problem.

5. IMPLEMENTATIONS

5.1 Hardware-in-the-loop Testing of a Large-scale Autonomous Target-Task Assignment Problem for a UAV Network

As a first step of coordination algorithm implementation within the mission simulator, we considered one of the basic problems. The problem consists of *n* targets to be visited by m UAVs and the vehicles should autonomously find the waypoint selections that result in the minimum total path traveled by the UAV fleet. In addition, all the mission coordination and the communication between the units had to be done autonomously without any human intervention. To address the first step of this problem, we have developed a large-scale distributed task/target assignment method that allows autonomous and coordinated task-assignment in real-time for a UAV fleet. By using delayed column generation approach on the most primitive non-convex supply-demand formulation, a computationally tractable distributed coordination structure (i.e. a market created by the UAV fleet for tasks/targets) is exploited. This particular structure is solved via a fleet-optimal dual simplex ascent in which each UAV updates its respective flight plan costs with a linear update of waypoint task values as evaluated by the market. The complete theoretical treatment and algorithmic structure of this problem can be found in [20].

5.2 Expansion of the Human-Machine-Interface (HMI) to decision-support system for manned and unmanned fleets

The development of agile C2 system for UAV fleets is very hard task, since the agility characteristics of a such system itself includes several key dimensions which should be examined in very different domains and context: robustness, resilience, responsiveness, flexibility, innovation, adaptation [4]. In this application, we focus on development of decision-support tools in order to improve the agility of C2 system in the sense of responsiveness, since the problem of high level UAV fleet coordination and the underlying low-level missions are performed in a very high tempo.

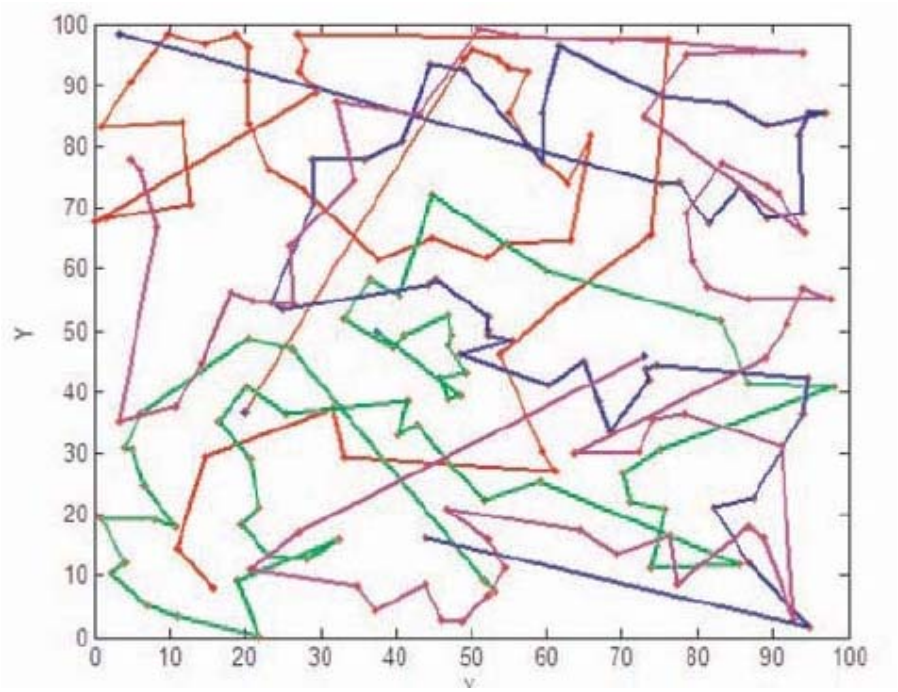

Figure 5.1 : The waypoint routes for a random pop-up task-target assignment for four vehicles. The algorithm is implemented further in the receeding horizon mode for five hundred waypoints.

A key part of such an event-driven process interaction hinges on the UAVs to autonomously coordinate the target/task assignments and distributions. The experimental illustration of this within the mission simulator is illustrated in [34]. However, it is important to note that supervisory control and interruption is desired in all mission critical phases. Towards this goal, we have developed Human-Machine-Interface display systems which allow this supervisory control capability over unmanned vehicle networks over the manned simulation platform. This HMI display systems enhanced with decision support capability generate candidate schedule which has temporal flexibility as much as possible and is robust to executional uncertainty in mission environment in a very brief time intervals. Then, the human operator examines the proposed schedule and makes modifications or posts additional precedence constraints between activities if required. Generation of robust schedules is very crucial, since the mission environment is inherently very challenging and includes too many misfortune like asset breakdowns or delays in activity duration. The developed algorithm has the capability of adapting the current solution into the new situation very fast; hence, it contributes the human operator to handle with the perturbation in the mission environment. In addition to showing

manned vehicle flight information data, this HMI display system also tracks and monitors the action of unmanned vehicle fleet within the joint mission.

Figure 5.2 : Command and Control Interface GUI

6. CONCLUSIONS

In this work, we have focused on the problem of the planning and scheduling of tasks in UAV fleet C2 applications and decision support architecture is presented for real-time decision generation for operators who are responsible for high-level decision making in scenarios involving a large number of tasks across multiple UAVs. The main segments of the proposed decision support architecture and its integration to a labscale mission simulator is explained in detail. Finally, the developed Human Machine Interfaces and the performance of the implemented algorithms for scheduling and C2 applications is given. In addition, an experimental mission simulator has been developed for joint simulation of manned and unmanned vehicle fleets at the same time. This platform serves not only for testing of distributed command and control algorithms and interfaces, but also provides real device hardware-in-the-loop test and real flight test vehicle integration capabilities within simulated scenarios. As such, it provides a realistic test platform to demonstrate and validate research on C2 algorithms and enabling technologies for joint manned-unmanned operations. Current research on this platform is focused on modifying hardware and algorithms with STANAG compliant devices and message formats, adding speech command capability for human operator and pilots, and providing augmented reality concept to the SVS screen by integration of real time streaming video from real unmanned vehicles flying in simulated scenarios.

REFERENCES

[1] **Nehme, C., Cummings, M. L. and Crandall, J.,** 2006. *UAV Mission Hierarchy.* Boston : Humans and Automation Laboratory, Massachusetts Institute of Technology.

[2] **Bruni, S., Mercier, S. and Mitchell, P. J.,** 2007. *Automation Architecture for Single Operator, Multiple UAV Command and Control*, The International C2 Journal, pp. 1-24.

[3] **Cummings, M. and Guerlain, S.,** 2004. *Human performance issues in supervisory control of autonomous airborne vehicles*, AUVSI Unmanned Systems North America Conference.

[4] **Alberts, D.S. and Hayes, R.E.,** 2003. *Power to the edge: command, control in the information age.* s.l. : Command and Control Research Program (CCRP) Publication Series.

[5] **McLain, T.W. and Beard, R.W.,** *Coordination Variables, Coordination Functions, and Cooperative Timing Missions*, Journal of Guidance, Control & Dynamics, Vol. 28, pp. 150-161, 2005.

[6] **ParunaK, H.V.D. and Brueckner, S. and Sauter, J.,** 2002. *Digital pheromone mechanisms for coordination of unmanned vehicles,* New York : ACM Press, Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1. pp. 449-450.

[7] **Godbole, D. N.,** 1999. *Control and coordination in uninhabited combat air vehicles* American Control Conference, Vol. 2, 1999

[8] **Sastry, S. S.,** 1999. Hybrid Control Synthesis Real-Time Control Problems for UAV. s.l. : University of California, Berkeley.

[9] **Makarenko, A. A.,** 2004. A decentralized Architecture for Active Sensor Networks. *Phd Thesis.* Sydney : The University of Sydney.

[10] **Valenti, M., Schouwenaarsy, T., Kuwataz, Y., Feron, E. and How, J.,** 2004. *Implementation of a Manned Vehicle - UAV Mission System.* s.l. : AIAA Guidance, Navigation, and Control Conference and Exhibit.

[11] **Christiansen, R. S.,** 2004. Design of an Autopilot for Small Unmanned Aerial Vehicles. *MSci Thesis.* s.l. : Brigham Young University.

[12] **OpenSource, GPL.** Boost C++ Libraries. [Online] 2001. http://www.boost.org/.

[13] **Jones, E. D., Roberts, R. S. and Hsia, T. C. S.,** 2003. *STOMP: a software architecture for the design and simulation of UAV-based sensor networks.* Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on. Vol. 3, pp. 3321-3326.

[14] **Cummings, M. L., Bruni, S., Mercier,S. and Mitchell, P. J.,** 2007. *Automation Architecture for Single Operator, Multiple UAV*

Command and Control. The International Command and Control Journal, pp. 1-2.

[15] **Policella, N., Smith, S. F., Cesta, A. and Oddi, A.,** 2004. *Generating Robust Schedules through Temporal Flexibility.* Proceedings of the 14th International Conference on Automated Planning and Scheduling.

[16] **Policella, N., Oddi, A., Smith, S. F. and Cesta, A.,** 2004. *Generating Robust Partial Order Schedules.* s.l. : Springer, Lecture Notes in Computer Science, pp. 496-511.

[17] **Cummings, M. L. and Guerlain, S.,** 2004. *An Interactive Decision Support Tool for Real-time In-flight Replanning of Autonomous Vehicles.* AIAA 3 rd" Unmanned Unlimited" Technical Conference, Workshop and Exhibit, pp. 1-8.

[18] **Smith, S. F.,** 2003. *Is Scheduling a Solved Problem?* s.l. : Springer, Proceedings First Multi-Disciplinary International Conference on Scheduling: Theory and Applications (MISTA 03).

[19] **Policella, N.,** 2005. Scheduling with uncertainty: A proactive approach using Partial Order Schedules. *Phd Thesis.* Rome : University of Rome ``La Sapienza".

[20] **Karaman, S. and Inalhan, G.,** 2008. *Large-scale Task/Target Assignment for UAV Fleets Using a Distributed Branch and Price Optimization Scheme.* Seoul, South Korea : s.n., Int. Federation of Automatic Control World Congress (IFAC WC'08).

[21] **Bartusch, M., Möhring, R. H. and Radermacher, F. J.,** 1988. *Scheduling project networks with resource constraints and time windows.* s.l. : Springer, Annals of Operations Research, Vol. 16, pp. 199-240.

[22] **Patterson, J. H., Slowinski, R., Talbot, F. B. and Weglarz, J.,** 1989. *An algorithm for a general class of precedence and resource constrained scheduling problems.* s.l. : Amsterdam, Elsevier Publisher, Advances in Project Scheduling, Vol. 28.

[23] **Patterson, J. H., Talbot, F. B., Slowinski, R. and Weglarz, J.,** 1990. *Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling problems.* European Jour. of Operational Research, Vol. 49, pp. 68-79.

[24] **Sprecher, A.,** 1994. *Resource-constrained project scheduling - exact methods for the multi-mode case.* Berlin : Springer.

[25] **Sadeh, N.,** 1991. *Look-ahead techniques for micro-opportunistic job shop scheduling.* Pittsburg : Carnegie Mellon University, CS91-102.

[26] **Nuijten, W. and Le Pape, C.,** 1998. *Constraint-Based Job Shop Scheduling with IILOG Scheduler.* s.l. : Springer, Journal of Heuristics, Vol. 3, pp. 271-286.

[27] **Smith, S. F. and Cheng, C. C.,** 1993. *Slack-Based Heuristics for Constraint Satisfaction Scheduling.* s.l. : John Wiley & Sons Ltd., Procedings of the National Conference on Artificial Intelligence, p. 139.

[28] **Cheng, C. C. and Smith, S. F.,** 1994. *Generating feasible schedules under complex metric constraints.* s.l. : American Association for Artificial Intelligence Menlo Park, CA, USA, Proceedings of the 12th national conference on Artificial intelligence (vol. 2) table of contents, pp. 1086-1091.

[29] **Cesta, A. and Oddi, A. and Smith, S. F.,** 1999. *An Iterative Sampling Procedure for Resource Constrained Project Scheduling with Time Windows.* s.l. : Lawrence Erlbaum Associates Ltd., International Joint Conference On Artificial Intelligence, Vol. 16, pp. 1022-1031.

[30] **Cesta, A., Oddi, A. and Smith, S. F.,** 1998. *Profile-based algorithms to solve multiple capacitated metric scheduling problems.* s.l. : AAAI Press, In Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98). pp. 214-223.

[31] **Dechter, R., Meiri, I. and Pearl, J.,** 1991. *Temporal constraint networks.* s.l. : Elsevier Science Publishers Ltd. Essex, UK, Artificial Intelligence, Vol. 49, pp. 61-95.

[32] **Kolisch, R. and Schwindt, C. and Sprecher, A.,** 1999. *Benchmark Instances for Project Scheduling Problems.* s.l. : Kluwer Academic Publishers, Project Scheduling: Recent Models, Algorithms, and Applications.

[33] **Schwindt, C.,** 2008. Project Generator ProGen/max and PSP/max-library. [Online] Universität Karlsruhe (TH) - Institute for Economic Theory and Operations Research, http://www.wior.uni-karlsruhe.de/LSNeumann/Forschung/ProGenMax/rcpspmax.html.

[34] **Arslan, O., Armagan, B. and Inalhan, G.** Development of a Mission Simulator for design and testing of C2 Algorithms and HMI Concepts across Real and Virtual Manned-Unmanned Fleets. *Optimization and Cooperative Control Strategies.* s.l. : Springer, 2008.

APPENDICES

APPENDIX A.1

Algorithm 1 Continuous Update Loop

```
Data : dataID
RoutingData : temp
continuousThreadRunning[dataID]←true
while isStarted is true do
    dataPacket ← GetNextPacket()
    dataID ← dataPacket.dataID
    buffer[dataID] ← dataPacket.data
    if ContinuousRoutingTable[dataID] is not empty then
        while continuousRoutingTable[dataID] is not ended do
            routeData ← continuousRoutingTable[dataID].Read()
            intID ← routeData.interfaceID
            unit ← routeData.unit
            data ← dataPacket.data
            dataLength ← sizeof(data)
            interface[intID].Send(data, dataLength, unit)
        end while
    end if
end while
continuousThreadRunnig[dataID] false
```

APPENDIX A.2

Algorithm 2 Periodic Update Loop

```
Require: Data dataID
    timer ← 0
    counter ← 0
    RoutingData : routeData
    TimingInfo : timeValue
    periodicThreadRunning[dataID] ← true
    while isStartedPeriodic[dataID] is true do
        if periodicRoutingTable[dataID] is not empty then
            while periodicRoutingTable[dataID] is not ended do
                routeData ← periodicRoutingTable[dataID].Read()
                timeValue ← time[dataID].Read()
             dataIntPeriod ← routeData.period/timeValue.realSampling
                if counter is multiply of dataIntPeriod then
                 intID ← routeData.interfaceID
                 unit ← routeData.unit
                 data ← buffer[dataID]
                 dataLength ← sizeof(data)
                 interface[intID].Send(data, dataLength, unit)
                end if
            end while
            timeValue ← time[dataID]
            timer ← timer + timeValue.realSampling
            counter ← timer/timeValue.realSampling
            intPeriod ← timeValue.readPeriod/timeValue.realSampling
            counter ← counter (mod intPeriod)
            sleep(timeValue.realSampling)
        end if
    end while
    periodicThreadRunning[dataID] ← false
```

APPENDIX A.3

Algorithm 3 $ESTA_P(P, h_{max})$

Input : a problem P and horizon of the problem h_{max}

Output : a solution S

1. $S^P \leftarrow CreateCSP(P)$
2. $S^P \leftarrow S^P \cup \{t_e - t_s \leq h_{max}\}$
3. $S^P \leftarrow ESTA_{PR}(S^P, h_{max})$
4. **if** Exists failure in S^P **then**
5. **return** FAILURE
6. **else**
7. $S \leftarrow ComputeESS(S^P)$
8. **end if**
9. **return** S

APPENDIX A.4

Algorithm 4 LevelResourceDemand($S^P, n \leftarrow \infty$)

Input : a partial solution of Problem S^P

Output : a partial solution of problem S^P

1. **for** $i = 1$ to n **do**
2. $C_S \leftarrow$Select-Conflict-Set(S^P)
3. **if** $C_S = \emptyset$ **then**
4. **return** S^P
5. **else if** Exist an unresolvable conflict in C_S **then**
6. **return** FAILURE
7. **else**
8. $(a_i \prec a_j) \leftarrow$Select-Leveling-Constraint(C_S)
9. $S^P \leftarrow S^P \cup (a_i \prec a_j)$
10. **end if**
11. **end for**
12. **return** S^P

APPENDIX A.5

Algorithm 5 $ESTA_{PR}(S^P, h_{max})$

Input : a partial solution of problem S^P and horizon of the problem h_{max}

Output : a partial solution S^P

1. **if** size(P) < minimum size **then**
2. $S^P \leftarrow LevelResourceDemand(S^P)$
3. **return** S^P
4. **else**
5. **loop**
6. **loop**
7. $t_c \leftarrow CreatePartition(S^P)$
8. $S^{P_1} \leftarrow \forall a_i | s_{a_i} \in P \wedge est(s_{a_i}) < t_c$
9. **if** $0 < size(P_1) < size(P)$ **then**
10. **break**
11. **end if**
12. $S^P \leftarrow LevelResourceDemand(S^P, 1)$
13. **end loop**
14. $S_0^P \leftarrow S^P$
15. $P_1 \leftarrow P_1 \cup t_{de}$
16. $S^{P_1} \leftarrow S^{P_1} \cup \{t_{de} - t_s \leq h_{max}\}$
17. $S^{P_1} \leftarrow ESTA_{PR}(S^{P_1}, h_{max})$
18. **if** failure exists in S^{P_1} **then**
19. **return** FAILURE
20. **else**
21. $S^{P_2} \leftarrow \forall a_i | e_{a_i} \in P \wedge est(e_{a_i}) > t_c$
22. $P_2 \leftarrow P_2 \cup t_{ds}$
23. $S^{P_2} \leftarrow S^{P_2} \cup \{t_e - t_{ds} \leq h_{max}\}$
24. $S^{P_2} \leftarrow ESTA_{PR}(S^{P_2}, h_{max})$
25. **if** failure exists in S^{P_2} **then**
26. **return** FAILURE
27. **else**
28. $S^P \leftarrow S^{P_1} \cup S^{P_2}$
29. **if** S^P is time feasible **then**
30. **return** S^P
31. **end if**
32. **enf if**
33. **end if**
34. $S^P \leftarrow S_0^P$
35. $S^P \leftarrow LevelResourceDemand(S^P, 1)$
36. **end loop**
37. **end if**

APPENDIX A.6

Algorithm 6 CreatePartition(S^P)

Input : a partial solution of problem (S^P)

Output : a time point t_c

1. $n_1 \leftarrow 0$, $n_2 \leftarrow 0$
2. **for all** time point $t_j \in P$ **do**
3. **for all** activity $a_k \in P$ **do**
4. **if** $est(s_{a_k}) < est(t_j)$ **then**
5. $n_1 \leftarrow n_1 + 1$
6. **end if**
7. **if** $d(t_j, e_{a_k}) > 0$ **then**
8. $n_2 \leftarrow n_2 + \frac{d(t_j, e_{a_k}) + \min(d(e_{a_k}, t_j), 0)}{d(t_j, e_{a_k}) + d(e_{a_k}, t_j)}$
9. **end if**
10. **end for**
11. $c_j \leftarrow Cost(n_1, n_2, n)$
12. **end for**
13. $t_c \leftarrow t_j | c_j = \min_{t_i}(c_i)$
14. **return** $est(t_c)$

CURRICULUM VITA

Candidate's full name: Oktay ARSLAN

Place and date of birth: Sariyer 10/07/1984

Permanent Address: Mehmet Akif Mah. 1. Nergis Sok. No: 15 D1 Ikitelli Istanbul Türkiye

Universities and Colleges attended:

Istanbul Technical University, Istanbul, Turkey Sep. 2007 – Jun. 2009
MSci. in Defense Technologies Program - Information Technology Option

Istanbul Technical University, Istanbul, Turkey Sep. 2003 – Aug. 2007
BSci. in Computer Engineering Program (Double major)

Istanbul Technical University, Istanbul, Turkey Sep. 2002 – Jun. 2006
BSci. in Electrical Engineering Program - Control Engineering Option

Publications:

- **Arslan, O.** and İnalhan, G., Design of a Decision Support Architecture for Human Operators in UAV Fleet C2 Applications. *14th International Command and Control Research and Technology Symposium,* June 15-17, 2009 Washington, DC, USA

- **Arslan, O.** and İnalhan, G., An Event Driven Decision Support Algorithm for Command and Control of UAV Fleets. *Amercican Control Conference 2009,* June 10-12, 2009 St. Louis, Missouri, USA

- **Arslan, O.**, Armağan, B. and İnalhan, G. , Development of a Mission Simulator for design and testing of C2 Algorithms and HMI Concepts across Real and Virtual Manned-Unmanned Fleet. *Optimization and Cooperative Control Strategies, Volume 381/2009, Lecture Notes in Control and Information Sciences,* October 18, 2008 Springer Berlin

- **Arslan, O.**, Ulualan, B. G. and İnalhan, G., Design and Implementation of Communication and Information Distribution Modules for Cooperative Unmanned-Manned Vehicle Networks. *8th International Conference on Cooperative Control and Optimization* January 30 – February 1, 2008 Gainesville, Florida, USA

- Çetinkaya, A., Karaman, S. **Arslan, O.**, Aksugür, M. And İnalhan, G., Design of a Distributed C2 Architecture for Interoperable Manned/Unmanned Fleets. *7th International Conference on Cooperative Control and Optimization* January 31 – February 2, 2007 Gainesville, Florida, USA