NAVAL SURFACE TACTICAL MANEUVERING SIMULATION SYSTEM (NSTMSS)

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

OKAN TOPÇU

90662

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF COMPUTER ENGINEERING

T.C. YÜKSEKÖĞRETİM KURULU
DOKÜMANTASYON MERKEZİ

DECEMBER 1999

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Tayfur ÖZTÜRK
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science

Prof. Dr. Batoş Yarman VURAL
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science

Asst.Prof.Dr. Halit OĞUZTÜZÜN
Supervisor

Examining Committee Members:

Prof. Dr. İbrahim AKMAN
Assoc. Prof. Dr. Ali DOĞRU
Assoc. Prof. Dr. Veysi İŞLER
Asst. Prof. Dr. Halit OĞUZTÜZÜN
Ltd. Cdr. Kemal SALAR, M.S.

ABSTRACT

NAVAL SURFACE TACTICAL MANEUVERING SIMULATION SYSTEM (NSTMSS)

This thesis introduces a Distributed Interactive Simulation (DIS) application called "Naval Surface Tactical Maneuvering Simulation System (NSTMSS)", and describes its architecture and implementation. NSTMSS is a virtual simulation system that is composed of a three dimensional ship handling simulator, a tactical level simulation of operational area, and a virtual environment for the ship simulators.

NSTMSS has been developed by using the concepts of High Level Architecture (HLA), which provides a structural basis for interoperability and reusability. NSTMSS uses Runtime Infrastructure (RTI) for data communication and object exchange, Performer for 3D graphical interfaces and virtual environment. Unified Modeling Language (UML) has been used for Object Oriented Analysis and Design (OOAD).

NSTMSS provides a Networked Virtual Environment (NVE) testbed for naval surface actions in which new tactics, operations and formations can be evaluated and tested as well as the present ones can be analyzed.

Keywords: DIS, FEDEP, HLA, Naval Simulation, Networked Virtual Environment, Object Oriented Analysis & Design, RTI, UML

ÖZ

SUÜSTÜ GEMİLERİ TAKTİK MANEVRA SİMÜLASYON SİSTEMİ (NSTMSS)

Bu tez, bir Dağıtık Etkileşimli Simülasyon (DES) uygulaması olan Suüstü Gemileri Taktik Manevra Simülasyon Sistemini (NSTMSS), onun yapısını ve gerçekleştirimini anlatmaktadır. NSTMSS, bir gemi manevra simülatörünü, bu gemilerin manevra yapacağı sanal ortamın oluşturulmasını, ve harekat sahasının taktik seviyedeki simülasyonunu kapsayan bir sanal simülasyon sistemidir.

NSTMSS, tekrar kullanılabilirlilik ve karşılıklı işlerlik için yapısal bir mimari sağlayan Yüksek Seviye Mimarisi konsepti kullanılarak geliştirilmiştir. Bu bağlamda NSTMSS; veri haberleşmesi ve nesne değişimi için Run Time Infrastructure fonksiyonlarını, üç boyutlu grafik arayüzü ve sanal ortam için Performer grafik kütüphanesini, Nesne yönemli analiz ve tasarım için Unified Modeling Language (UML) kullanmaktadır.

NSTMSS, suüstü harekatı için, mevcut ve yeni taktiklerin, harekatların, ve nizamların test edilip denenebileceği bir Ağlandırılmış Sanal Ortam için test ortamı olarak hizmet etmektedir.

Anahtar kelimeler: Dağıtık Etkileşimli Simülasyon, Deniz Simülasyonu, Ağlandırılmış Sanal Ortam, Nesne Yönelimli Analiz ve Tasarım, RTI, UML

To My Family

ACKNOWLEDGMENTS

I express sincere appreciation to Asst. Prof. Dr. Halit Oğuztüzün for his guidance and insight throughout the research.

Thanks go to Turkish Navy, which supported this thesis by providing leave of absence for the author.

This work has been partially supported by Middle East Technical University grant AFP-97-07-04-01.

TABLE OF CONTENTS

ABSTRACT iii
ÖZ iv
ACKNOWLEDGMENTS vi
TABLE OF CONTENTS vii
LIST OF FIGURES xii
LIST OF TABLES xiv
LIST OF ABBREVIATIONS xv

CHAPTER
1. INTRODUCTION
1.1. GENERAL PROBLEM DEFINITON 2
1.2. RATIONALE FOR NSTMSS 5
1.3. OBJECTIVES AND SCOPE 7
1.4. TECHNOLOGY OVERVIEW 11
1.4.1. Distributed Interactive Simulation (DIS) 11
1.4.2. High Level Architecture (HLA) 13
1.4.3. Unified Modeling Language (UML) 19
1.5. TOOLS 21
1.5.1. Performer 21
1.5.2. Run-Time Infrastructure (RTI) 23
1.5.3. Rational Rose 25
1.5.4. Object Model Development Tool (OMDT) 27
1.6. RELATED WORK 28
1.6.1. NPSNET 28
1.6.2. Full Mission Shiphandling Bridge Simulator 28
1.7. SUMMARY OF CHAPTERS/THESIS OUTLINE 29

2. NSTMSS ARCHITECTURE
2.1. SYSTEM COMPONENTS 31
2.2. DESIGN PHILOSOPHY 32
2.3. FEDERATION DESIGN 33
2.3.1. RTI Provided Processes 35
2.3.2. Publish/Subscribe (P/S) Pattern 35

2.4. INITIALIZATION AND SHUT-DOWN PRINCIPLES 41
2.5. NSTMSS COORDINATE SYSTEM 41
2.6. TIME MANAGEMENT 47
2.7. SUMMARY 48

3. NSTMSS FEDERATES
3.1. KNOX FEDERATE (KNOXFd) 49
3.1.1. KNOXFd Modular Structure 50
3.1.2. Graphical User Interface 55
3.1.3. Simulation of Bridge Equipment and Indicators 60
3.1.4. Hydrodynamics and Maneuvering 62
3.1.5. Basic Program Flow 65
3.1.6. Scene Construction 69
3.1.7. Environmental Effects 71
3.2. OTC FEDERATE (OTCFd) 71
3.2.1. OTCFd Modular Structure 71
3.2.2. Graphical User Interface 73
3.2.3. Tactical Message System (TMS) Modeling for Federation 74
3.3. ENVIRONMENT FEDERATE (EnviFd) 79
3.4. FEDERATION MANAGEMENT FEDERATES 79
3.4.1. OSE Federate (OSEFd) 80
3.4.2. Federation Monitor Federate 81

4. HARDWARE ARCHITECTURE AND NETWORK ISSUES
4.1. EXPERIMENTAL HARDWARE CONFIGURATION 82
4.2. TECHNICAL ENVIRONMENT 83
4.3. FEDERATION EXECUTION PLANNER'S WORKBOOK 84

5. DEVELOPMENT METHODOLOGY
5.1. FEDERATION DEVELOPMENT AND EXECUTION PROCESS (FEDEP) 89
5.2. OBJECT ORIENTED ANALYSIS AND DESIGN (OOA&D) 92
5.3. CONCLUSIONS AND ASSESSMENT 93

6. DISCUSSION AND CONCLUSIONS
6.1. CONTRIBUTIONS OF THIS WORK 96
6.2. SUGGESTIONS FOR FUTURE WORK 98

APPENDICES

APPENDIX – A

SYSTEM REQUIREMENTS ANALYSIS

A.1 SOFTWARE REQUIREMENTS 102
A.1.1 General Requirements 102
A.1.2 STG Requirements 105
A.1.3 TMS Requirements 105
A.1.4 OOD Requirements 108
A.1.5 OTC Requirements 109
A.1.6 OSE/Scenario Monitor Requirements 109
A.1.7 Federation Monitor (FEDMON) Requirements 110
A.1.8 Navigation Radar Requirements 110
A.1.9 Helm Requirements 111
A.1.10 Engine Order Telegraph (EOT) Requirements 111
A.1.11 GPS Requirements 111
A.1.12 Nautical GIS Requirements 111
A.1.13 Navigation Chart Requirements 112
A.1.14 Formation Requirements 112
A.2 HARDWARE REQUIREMENTS 117
A.2.1 General Requirements 117

APPENDIX – B

OBJECT ORIENTED ANALYSIS (OOA)

B.1 SYSTEM USERS/ACTORS 118
B.2 USE CASE DIAGRAMS 119
B.2.1 Use Case Identification 120
B.2.2 Top Level "OOD" Use Case Diagram 120
B.2.3 Top Level "OTC" Use Case Diagram 121
B.3 INTERACTION DIAGRAMS 122
B.3.1 "OOD Ship handling" Interactions 124
B.3.2 "OOD Message handling" Interactions 136
B.3.3 "OTC STG handling" Interactions 139
B.3.4 "OTC Message handling" Interactions 142

APPENDIX – C

OBJECT ORIENTED DESIGN (OOD)

C.1 CLASS DIAGRAMS 143
C.1.1 Knox Federate (KNOXFd) Class Diagrams 143
C.1.2 OTC Federate Class (OTCFd) Diagrams 147
C.1.3 Environment Federate Class (EnviFd) Diagrams 149

APPENDIX – D
IMPLEMENTATION DETAILS
D.1 DEAD RECKONING ALGORITHM 155
D.2 CALCULATION OF A TRUE BEARING OF A RADAR ECHO 156
D.3 EnviFd COMMAND LINE INTERFACE 158

APPENDIX – E
PRACTICAL MATTERS
E.1 FEDEP ISSUES 160
E.2 RTI ISSUES 160
E.3 FED FILE ISSUES 162
E.4 C++ ISSUES 163
E.5 A FEW WORDS ABOUT CASE TOOL ROSE 163

APPENDIX – F
A NOTATION FOR FEDERATION DESIGN
F.1 MOTIVATION 165
F.2 NOTATION 167
F.2.1 Architectural Level Federation Design 167
F.2.2 HLA Classes 169
F.2.3 Publish/Subscribe Diagrams 170

APPENDIX – G
OBJECT MODEL TEMPLATE for NSTMSS
G.1 FEDERATION OBJECT MODEL (FOM) 174
G.2 SIMULATION OBJECT MODEL (SOM) 175
G.3 HLA OMT COMPONENTS 175
G.4 NSTMSS FOM DOCUMENTATION 176
G.4.1 Object Model Identification Table 178
G.4.2 Object Class Structure Table 178
G.4.3 Interaction Class Structure Table 178
G.4.4 Attribute Table 179
G.4.5 Parameter Table 180
G.4.6 Enumerated Datatype Table 180
G.4.7 Complex Datatype Table 184
G.4.8 Routing Space Table 185
G.4.9 FOM Lexicon 185
G.5 KNOX FEDERATE SOM DOCUMENTATION 187
G.6 OTC FEDERATE SOM DOCUMENTATION 193

T.C. YÜKSEKÖĞRETİM KURULU
DOKÜMANTASYON MERKEZİ

APPENDIX – H
GLOSSARY 200

REFERENCES 203
SELECTED WWW SITES 206

LIST OF FIGURES

1.1 Formation Example 3
1.2 Turn Example 4
1.3 Object Model Template Contents 15
1.4 Database Hierarchy Example of A Performer Scene 22
1.5 RTI Components At-a-glance 23
1.6 Federation Execution Lifecycle 24
1.7 Rose User Interface 26
1.8 OMDT User Interface 27

2.1 NSTMSS Components 32
2.2 Architectural Overview of NSTMSS 34
2.3 General P/S Diagram of NSTMSS Federation 37
2.4 Environment Object P/S Diagram 38
2.5 OTC Object P/S Diagram 39
2.6 Ship Object P/S Diagram 40
2.7 Communication Interaction P/S Diagram 41
2.8 Typical Federation Execution 41
2.9 Basic Program Flow of Typical Federates 42
2.10 Geodetic Coordinate System 44
2.11 Geocentric Coordinate System 45
2.12 Marcator Projection 46

3.1 KNOXFd Modular Structure 50
3.2 Simulation-Oriented Modules Internal Structure 51
3.3 Repository Connections 53
3.4 Frigate and sGUI Class Diagram 54
3.5 KNOXFd User Interface 57
3.6 The OOD's Positions 57
3.7 Views from (a) the Astern Deck (b) the Bow 58
3.8 The Command Toolbar Interface (CTI) 58
3.9 The Helm and EOT Panel 59
3.10 Message Interface 61
3.11 System Initialization 64

3.12 Main Simulation Loop 66
3.13 System Termination 68
3.14 How Performer Renders a Scene Graph? 68
3.15 Main Scene Graph 70
3.16 OTCFd Modular Structure 71
3.17 OTC Interface 72
3.18 (a) Class Diagram 73
(b) Collaboration Diagram 73
3.19 Sending Message Sequence Diagram 74
3.20 NSTMSS Communication Regions 75
3.21 Message Center Scene Graph 77
3.22 OSEFd Modular Structure 79

4.1 Execution Environment 81

5.1 FEDEP Version1.2 89
5.2 Development Lifecycle of NSTMSS 94

A.1 Ahead Sector View of the Ship 107
A.2 Radar GUI 109
A.3 Compass, Rudder Indicator and Engine Order Telegraph GUI 110
A.4 True Bearing Formation Example 112
A.5 Relative Bearing Formation Example 113
A.6 Turn Example 113
A.7 S-Turn Example 114
A.8 Rule-13 (b) Example 114
A.9 Rule-14 (a) Example 115
A.10 Rule-17 (a)(b)(c) Example 116

B.1 System Users/Actors 118
B.2 Top Level OOD Use Case Diagram 120
B.3 Top Level OTC Use Case Diagram 121
B.4 Sequence Diagram for "Using the Helm" 124
B.5 Sequence Diagram for "" 125
B.6 Collaboration Diagram for "Updating Course Display" 125
B.7 Sequence Diagram for "Using the EOT" 127
B.8 Collaboration Diagram for "Using the EOT" 128
B.9 Sequence Diagram for "Looking the GPS display" 128
B.10 Collaboration Diagram for "Updating GPS Values" 129
B.11 Sequence Diagram for "Changing the Radar Scale" 130

B.12 Sequence Diagram for "Measuring the Distance of a Radar Echo" 131
B.13 Sequence Diagram for "Measuring the Bearing of a Radar Echo" 132
B.14 Sequence Diagram for "Selecting the Chart" 132
B.15 Sequence Diagram for "Drawing Routes" 133
B.16 Sequence Diagram for "Looking Ahead Sector" 133
B.17 Sequence Diagram for "Looking Right/Left Sector" 134
B.18 Sequence Diagram for "Looking Astern Sector" 134
B.19 Sequence Diagram for "Sending the Message" 136
B.20 Collaboration Diagram for "Sending Messages" 137
B.21 Sequence Diagram for "Seeing the Incoming Message" 137
B.22 Sequence Diagram for "Changing the Ship" 138
B.23 Sequence Diagram for "Scrolling the Display" 139
B.24 Sequence Diagram for "Zoom In" 139
B.25 Sequence Diagram for "Zoom Out" 140

C.1 KNOX Class Diagram 141
C.2 Well-Defined Interface Between Modules 142
C.3 Frigate and sGUI Class Hierarchy Diagram 143
C.4 Frigate Class Hierarchy (Cont.) 144
C.5 OTC Federate Class Diagram 146
C.6 Environment Federate Class Diagram 147

D.1 True Bearing 150
D.2 Mathematical Model 151

F.1 Federate Notation 162
F.2 Architectural Federation Design Example 163
F.3 Class Notation 164
F.4 Class Based P/S Diagram 165
F.5 P/S Diagram Example With Inherited Classes 166
F.6 Federate Based P/S Diagram example 167

LIST OF TABLES

4.1 Technical Environment for NSTMSS Federates 82
4.2 Federation Execution Summary 83
4.3 Federate Summary in Execution 83
4.4 Information About Hosts in Execution 84
4.5 LAN Description 84
4.6 KNOXFd RTI Services 85
4.7 OTCFd RTI Services 86
4.8 EnviFd RTI Services 87

A.1 Turning Dimensions 102
A.2 Engine Telegraph Commands 107

B.1 System Top Level Use Cases 119
B.2 OOD Ship Handling Interactions 123
B-3 OOD Message Handling Interactions 135
B-4 OTC STG Handling Interactions 138

F-1 Multiplicity 162

G.1 Object Model Identification Table 171
G.2 Object Class Structure Table 172
G.3 Interaction Class Structure Table 175
G.4 Attribute Table 175
G.5 Attribute Table (Cont.) 176
G.6 Parameter Table 177
G.7 Enumerated Datatype Table 178
G.8 Complex Datatypes 178
G.9 Routing Space Table 179
G.10 Object Class Definitions 180
G.11 Interaction Class Definitions 180
G.12 KNOXFd Object Model Identification Table 181
G.13 KNOXFd Object Class Structure Table 181
G.14 KNOXFd Interaction Class Structure Table 182

G.15 KNOXFd Complex Datatypes 182
G.16 KNOXFd Attribute Table 183
G.17 KNOXFd Parameter Table 185
G.18 KNOXFd Enumerated Datatype Table 186
G.19 KNOXFd Routing Space Table 186
G.20 KNOXFd Object Class Definitions 186
G.21 KNOXFd Interaction Class Definitions 187
G.22 OTCFd Object Model Identification Table 187
G.23 OTCFd Object Class Structure Table 188
G.24 OTCFd Interaction Class Structure Table 188
G.25 OTCFd Complex Datatypes 188
G.26 OTCFd Enumerated Datatype Table 189
G.27 OTCFd Routing Space Table 189
G.28 OTCFd Attribute Table 190
G.29 OTCFd Parameter Table 191
G.30 OTCFd Object Class Definitions 192
G.31 OTCFd Interaction Class Definitions 192

LIST OF ABBREVIATIONS

AIS: ALSP Infrastructure Software

ALSP: Aggregate Level Simulation Protocol

API: Application Programming Interface

ASW: Anti-Submarine Warfare

BR: Break

CASE: Computer-Aided Software Engineering

CMMS: Conceptual Model of Mission Space

CO: Commanding Officer

CORBA: Common Object Request Broker Architecture

CIWS: Close-In Weapons System

CPD: Critical Performance Data

Cpt.: Captain

CTI: Command Toolbar Interface

DARPA: U.S. Defense Advanced Research Projects Agency

DDM: Data Distribution Management

DIF: Data Interchange Format

DIS: Distributed Interactive Simulation

DMSO: Defense Modeling and Simulation Office

DNS: Digital Nautical Chart

DR: Dead Reckoning

EnviFd: ENVIronment FeDerate

FEDEP: FEderation Development and Execution Process

FEDMON: Federation Monitor

FEDMONFd: Federation Monitor Federate

FED: Federation Execution Data

FedExec: Federation Executive

FEPW: Federation Execution Planner's Workbook

FOM: Federation Object Model

GIS: Geographic Information Systems

GUI: Graphical User Interface

GPS: Global Positioning System

HF: High Frequency

HLA: High Level Architecture

Hp: Horsepower

IDL: Interface Definition Language

JOOD: Junior Officer Of the Deck

KNOXFd: KNOX FeDerate

Kts: Knots

LAT: Latitude

LibRTI: RTI Library

LONG: Longitude

Mk: Mark

MOM: Management Object Model

M&S: Modeling & Simulation

MSRR: Modeling and Simulation Resource Repository

Nm: Nautical Mile

NPSNET: Naval Postgraduate School Networked environment

NSTMSS: Naval Surface Tactical Maneuvering Simulation System (pronounced as *'NiSTMiSS'*)

NVE: Networked Virtual Environment

OMDT: Object Model Development Tool

OML: Object Model Library

OMT: Object Model Template/Object Modeling Technique

OOA: Object Oriented Analysis

OOAD: Object Oriented Analysis & Design

OOD: Officer Of the Deck/Object Oriented Design

OOSE: Object Oriented SW Engineering

OSE: Officer Scheduling the Exercise

OSEFd: OSE FeDerate

OTC: Officer in Tactical Command

OTCFd: OTC FeDerate

PDU: Protocol Data Unit

RID: Rudder Indicator Degree/RTI Initialization Data

RTI: RunTime Infrastructure

RtiExec: RTI Executive

SGUI: Single Graphical User Interface

SIMNET: Simulator Networking

SOM: Simulation Object Model

SSM: Surface-to-Surface Missile

STG: Surface Task Group

STGC: STG Commander

TASKORD: Task Order

TCG: Turkiye Cumhuriyeti Gemisi

TMS: Tactical Message System

TSS: Tracking Subsystem

UHF: Ultra High Frequency

UI: User Interface

UML: Unified Modeling Language

Yds: Yards

T.C. YÜKSEKÖĞRETİM KURULU
DOKÜMANTASYON MERKEZİ

CHAPTER 1

INTRODUCTION

Computers have been used to generate interactive simulated virtual environments-usually for training or entertainment-since the 1960s. Many special purpose approaches to visual and interactive simulation have been tried. For example, the NASA Kennedy Space Center newspaper *Spaceport News* described the Apollo7 astronaut training interactive simulator this way on March 28, 1968[1]:

"*Each simulator consists of an instructor's station, crew station, computer complex, and projectors to simulate the stages of a flight. Engineers serve as instructors, instruments keeping them informed at all times of what the pilot is doing. Through the windows, infinity optics equipment duplicates the scenery of space. The main components of a typical visual display for each window includes a 71-centimeter fiber-plastic celestial sphere embedded with 966 ball bearings of various sizes to represent the stars from the first through fifth magnitudes, a mission-effects projector to provide earth and lunar scenes, and a rendezvous and docking projector, which functions as a realistic target during maneuvers.*"

Interactive simulation systems have advanced significantly due to advances in hardware and software, and to a greater understanding of human perceptions. We can identify two communication models for interactive simulation systems.

[1] This quote is taken from the "Performer Programmer's Guide".

Centralized:

One computer collects all of the data from different machines, stores and changes in some collection of data structures, and sends the results back out to each participating machine. Although this model allows the programmer to develop a simple data structure to store and handle the data, it is not scaleable. As more players enter the simulation, all the player-processes suffer, since the load of the centralized computer increases and they must wait increasingly longer to make a move.

Distributed:

An alternate more scaleable approach is to incorporate a distributed model. Here each host maintains its own complete, local copy of the information about participants as well as performing the rendering, computation and animation of objects. When a program makes changes to its own state, it sends the update data so that the other programs can update their copy of that state.

This thesis presents a study of the distributed model of interactive simulation. We developed a distributed interactive simulation application called "*Naval Surface Tactical Maneuvering Simulation System-NSTMSS*". Thesis covers all the facts of the development lifecycle.

1.1 GENERAL PROBLEM DEFINITION

In a surface operation, the *Surface Task Group (STG)* should operate in *Formations* defined by maneuvering and formation rules given in the *Tactical Publications*. The Surface Task Group is composed of surface ships (e.g. frigates, destroyers, etc) and commanded by the Officer in Tactical Command (OTC)(also called STG Commander (STGC) or Commodore) to accomplish a specific task. OTC gives an order to form one of these formations in accordance to Operation Phase, and the Officer Of the Decks (OODs) of ships must adjust their ship's course and speed to take the appropriate position and to construct the ordered formation as well as

keeping their positions in a formation, which is also known as *Formation Steaming*. All OODs, their first duty, should know the formation and maneuvering rules. Officer Scheduling the Exercise (OSE) will give the task and the operation details to STGC.

WHAT IS TACTICAL MANEUVERING AND FORMATION?

All the movements of the naval surface task group are organized according to the commands of the Commodore (OTC). These commands must be structured and carefully calculated to ensure a well-defined fleet formation in time. This is not trivial because the movements of the ships in the fleet are limited by some constraints including the avoidance of crash between ships and avoidance of landing. A formation example is given in the figure 1.1. Before the ships leave the port or anchorage, all the ships in the STG establish a communication network, which is called Tactical Communication Network where all the voice communication is being carried out by Communication Officers. One station, typically the communication center of the ship which has OTC onboard, is designated as the *Major Control Station*, which controls and operates all communication. All the tactical maneuvering and operations are being ordered via this tactical communication channels.

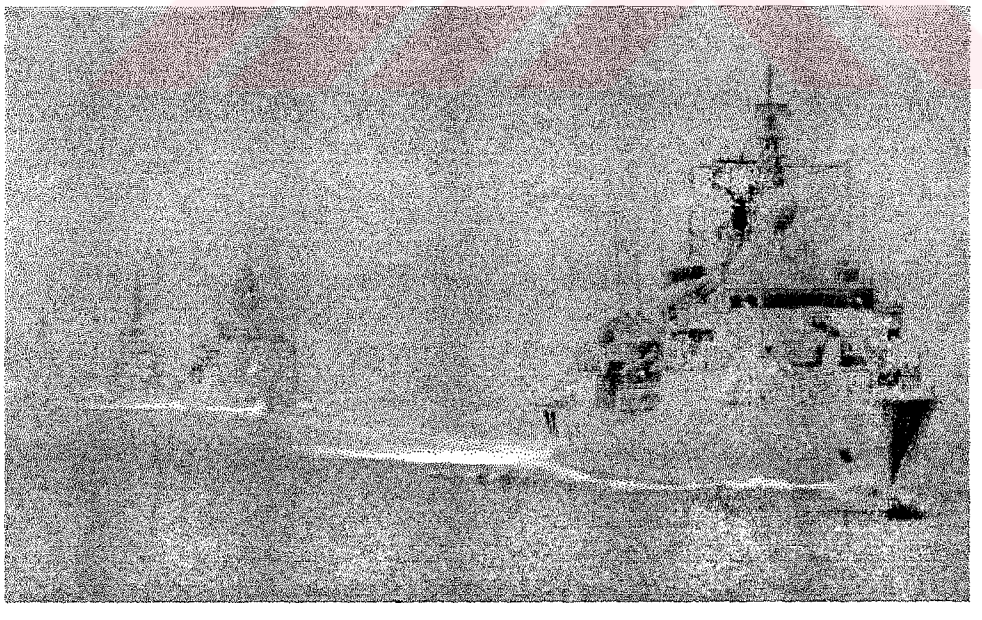


Figure 1.1 Formation Example

After establishing the tactical communication channel, the OTC gives his first order: "Leave the port. Departure formation is ....". OTC also orders which ship will leave the port first, what will be the formation speed and course, and the relative positions of the ships according to guide ship. The OODs of the ships construct the ordered formation as soon as possible, and once they succeed; they try to keep their positions in the formation.

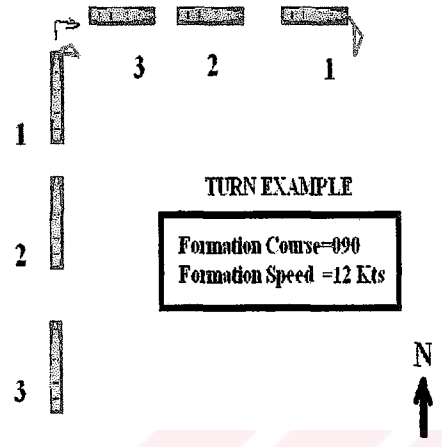


Figure 1.2 Turn Example

When OTC decides to order a new formation, he prepares a tactical message and sends it to all ships. As soon as the message is received by the ships, the Guide Ship starts to turn its ordered course and takes its position, formation course and speed, and informs all ships. The OODs of other ships determine their speeds and courses according to the guide ship and take their relative position according to guide ship.

This entire scenario describes "*Naval Surface Tactical Maneuvering*". For example, all the ships are moving to bearing 000 degrees, and OTC decided to change formation course to 090 degrees, as seen in the figure 1.2. The guide ship immediately turns to the new formation course and adjusts its speed to formation speed. The following ships take their correct positions relative to the guide.

While passing from one formation to another, the orders of the OOD play a very important role in the safety and security of the ships in the fleet. A wrong calculation may cause his ship not to have the correct position in the specified time or even cause his ship not to be able to form the formation at all. In the worst case a wrong calculation may result in accidents.

1.2 RATIONALE FOR NSTMSS

INCREASING DEMAND FOR JOINT TRAINING AND DISTRIBUTED SIMULATION:

Shiphandling, coupled with a true appreciation and understanding of the forces which act on one's vessel is a necessary skill, which all the naval watch officers must develop. A junior bridge officer can train in a stand-alone, high fidelity, full-scale ship simulator with the goal of developing shiphandling skills. However, the ability to perform a mission as an individual does not guarantee the ability to function as a member of a coordinated group. Especially for a naval surface operation, which is carried out by the participation of many ships, team training is crucial.

The OOD should develop two skills in parallel: the shiphandling skills and tactical maneuvering skills. The OOD can handle his ship very well when the ship is alone at sea. Also, you can train the OOD in a full mission stand-alone bridge simulator to make practice in shiphandling. But it is not enough; besides shiphandling skills, the OOD should quickly and correctly decide the course and speed of the ship in the formation, which means a joint operation. If you are in time, you are a good officer, else you should be trained. Of course, training is a long-term problem for the Navy.

Many simulation and training systems have been developed by different vendors. These monolithic simulation systems are used to train personnel to do their jobs as individuals or as members of a small team. Monolithic simulations cannot provide the full benefits of team training. To gain the full

benefits of team training in a virtual environment, several users from different levels must participate in the virtual environment. For example, in Turkish Navy, there are "*platform level training simulations*" like ship driving simulators and "*tactical level training simulations*" like naval surface warfare simulations; however these are used in separation.

Platform level training simulations are used to train the operators with the goal of learning one particular task, for example radar simulators to train the radar operators. On the other hand, tactical level training simulations provide bird's eye view of the operational area to depict the tactical picture for the operation commanders. For example, in a surface warfare simulation (or a war-game), there are red and blue forces. All entities in this picture are small icons over the operational area map.

THE RATIONALE FOR INTERFACING EXISTING SIMULATORS:

The simulation systems are *expensive* to develop, require large support organizations, and often specialize in specific battlefield domains. The vision is simple: "training thousands of entities in a common virtual environment from the whole domains to gain benefit of joint training". There are numerous simulation systems used in the Armed Forces specialized in specific domains. To catch up with the vision, the cheapest solution is to interface the existing simulation systems so that they all operate synchronously. All modern Navies try to link their existing shore-based human-in-the-loop training simulators to a tactical war-gaming system to provide a sophisticated command team training environment using DIS and/or HLA.

A QUESTION OF SAFETY:

Most Navy training was accomplished on board ships. This had many advantages. Young sailors and officers were exposed to the real-world environment at the appropriate time. When faced with limited underway time, the junior officer(s) may reap only limited benefit from the training opportunity at significant monetary cost. On the other hand, the concern for safe operations at sea has increasing importance to both commercial and Navy

interests. There have been a number of very costly maritime mishaps resulting in millions of dollars in lost revenues from ever increasing repair and cleanup cost.

As technology and war fighting became more complex, training migrated ashore, and an expensive and monolithic infrastructure evolved to provide trained personnel to the fleet.

Given today's computer technology, such systems could provide a number of very capable features and have near if not better "fidelity" than the full-scale mock-ups at a fraction of the cost - while providing far more increased access. These systems would significantly increase the availability of much needed training, and thus further aid in the development and enhancement of the situational awareness needed in developing "seaman's eye" so critically required in today's junior officer bridge watch standers.

1.3 OBJECTIVES AND SCOPE

This research project encompasses a number of objectives, which can be classified as *development objectives, principal objectives, and secondary objectives*.

Specifically *development objectives* were focused to develop extensible, multi-user, multi-platform, networked, and real-time processes from scratch with high-level 3D graphics properties in the areas:

- Design and implementation of a simple shiphandling simulator, plus modeling of some real-time hydrodynamic forces acting upon those ships within virtual environment,
- Design and construction of 3D virtual environment for those maritime entities,
- Design and implementation of plan-view of the operational area.

The *principal objectives* of NSTMSS are:

- To link platform and tactical level simulations to create a realistic virtual environment for the simulation of highly interactive naval surface operations. Multi-level training is the primary training goal for Navy's Modeling and Simulation (M&S) Vision. NSTMSS is combining platform and tactical level simulations to form distributed, man-in-the-loop, interactive simulation. We also transfer the problem of interfacing existing simulators by using HLA in lab environment.

- To provide reusability and interoperability. NSTMSS is based on HLA to achieve these goals. Interoperability and reusability are two key issues for the military and civilian Modeling and Simulation (M&S) Community. In this regard, the HLA as the newest technology for distributed simulation-based systems is considered a big step forward [D98].

- To gain experience with the HLA. This thesis addresses the phases of development of both parts, the HLA interface and the virtual simulator.

- To gain experience with the Federation Development And Execution Process (FEDEP) Model. The design and execution of a networked virtual environment are challenging tasks made even more difficult by the fact that DIS applications are becoming more complex and difficult to manage. In an HLA environment this includes the Federation Development and Execution Process (FEDEP) [DMSO98c].

- To form a testbed. HLA provides the network communication capabilities via the services of RTI. All the functionality of the recently proposed HLA and its components (RTI, rules and Object Model Template OMT) are being tested by the development of NSTMSS [DoD98a, DoD98b, and DoD98c]. NSTMSS can serve as a testbed:

- For interoperability and performance issues of HLA,
- For fine tuning RTI management service parameters, such as `tick()` function,
- For empirical evaluation of dead reckoning algorithms,
- For testing the efficiency of multicast protocols over a NVE,
- For equipment simulators, such as RADAR simulators.

❖ To gain the full benefits of team training in virtual environment. Like any other military operations, a naval surface operation involves planning, on time carrying out, and evaluation phases. A surface operation is being carried out by the participation of planning officers, ships and many navy-shore facilities. NSTMSS, which is a Distributed Interactive Simulation (DIS) application, should provide a training platform for the Officer Of the Decks (OODs) and Junior Officer Of the Decks (JOODs) to carry out tactical maneuvering exercises and the OTC to develop tactical picture and leadership. The System will also create an environment for naval surface actions which new tactics, operations and formations can be evaluated and tested as well as the present ones can be analyzed. The following users are presented in NSTMSS by giving the necessary tools for them with the goal in mind: "To gain the full benefits of team training in a virtual environment, several users from different levels must participate in.". The users (players) of NSTMSS are Officers of the Deck (OODs), Officer in Tactical Command (OTC), and Officer Scheduling the Exercise (OSE):

OFFICER OF the DECK (OOD):

The OOD is in overall charge of the safe operation of the ship. As such, he/she acts as a direct representative of the Commanding Officer (CO), handling the daily routine of the ship. OOD issues verbal orders to the helm to control the operation of the ship's rudder. He issues orders to the Engine Order Telegraph to control the ship's speed. The OOD receives all of his watch standing training "on the

job", usually by standing as Junior Officer Of the Deck (JOOD) under a more senior officer acting as OOD. This training typically continues until the JOOD achieves an acceptable level of proficiency and is qualified by the ship's CO.

OFFICER in TACTICAL COMMAND (OTC):

Surface Task Group (STG), which is composed of surface ships to achieve a specific task, is commanded by the Officer in Tactical Command (OTC). He is the present senior officer to whom has been delegated tactical command. He issues orders to ships to control the operation of STG. He is the principal responsible officer for performing the ordered task.

OFFICER SCHEDULING the EXERCISE (OSE):

OSE is the authority that defines the task, plans the operation or exercise, schedules the events (e.g. when STG will arrive harbor), also decides the participants (e.g. which ships will participate in STG, Who will be OTC). OSE also analyzes and reviews the operation or exercise in action or after-action.

There are some *secondary objectives*, which we may also call "*By-Products*". The intended scope of this project will include, but not be limited to, the following additional objectives:

- First, a reusable object library (C++ API), including both behavioral and polygonal models, is constructed in the course of developing NSTMSS.

- Second, Federation Development and Execution Process (FEDEP) methodology improvement is achieved by a graphical federation design notation and by iterative and incremental development approach.

This thesis study does not aim either 3D modeling techniques or graphic techniques (e.g. multiresolution, level of details (LOD), etc.).

1.4 TECHNOLOGY OVERVIEW

1.4.1 DISTRIBUTED INTERACTIVE SIMULATION (DIS)

Distributed Interactive Simulation involves a distributed virtual environment in which players communicate with others. In order that the databases don't lose accuracy, every agent in the virtual environment must be aware of every change in the environment. The primary mission of DIS is to define an infrastructure for linking simulations of various types at multiple locations to create realistic, complex, virtual worlds for the simulation of highly interactive activities. This infrastructure brings together systems built for separate purposes, technologies from different eras, products from various vendors, and platforms from various services and permits them to interoperate. DIS exercises are intended to support a mixture of virtual entities (man-in-the-loop simulators), live entities (platforms and test and evaluation systems), and constructive entities (wargames and other automated simulations). The DIS infrastructure provides interface standards, communication architectures, management structures, fidelity indices, technical forums, and other elements necessary to transform heterogeneous simulations into unified seamless synthetic environments.

DIS PROTOCOLS:

SIMNET (SIMULATION NETWORKING)

In 1985, the U.S. Defense Advanced Research Projects Agency (DARPA) sponsored the SIMNET programme to create a new standard for distributed interactive simulations [GM88]. SIMNET, not surprisingly, is oriented toward entities on a battlefield (tanks, planes, etc.), functional systems were available within three years of the release of the standard. SIMNET does not

require the use of specific network hardware or lower-level protocols, although it does define certain attributes such as minimum throughput and reliability. Ultimately, there were different implementations for SGI and Sun systems, but today it is not used [ZLPM92b].

DIS PROTOCOL:

DIS (IEEE1278) is a newer simulation standard than SIMNET. DIS protocol shares its goals and purposes with SIMNET, but is far more ambitious, allowing for greater complexity and realism (e.g. SIMNET uses a flat terrain, while DIS protocol accounts for the curvature of the earth.). The foundation of DIS is a standard set of messages and rules called *Protocol Data Units (PDUs)*, used for sending and receiving information across a computer network. PDUs are used to update remote variables and assure these variables have the same value at different locations of the simulation. Different types of the PDUs are used for different purposes. A set of PDUs are predefined; as further requirements arise, new PDUs should be defined and added to these standards. Every PDU has a format and entities cannot send messages, which do not fit given formats. DIS is strictly a peer-to-peer architecture, in which all data is transmitted to all simulators where it can be rejected or accepted depending on the receiver's needs [ZLPM92a].

DIS is these days maturing into a new generation of object request broker based software technology, known as the High Level Architecture (HLA). DIS community is adopting an object passing model rather than only exchanging PDUs.

ALSP (AGGREGATE LEVEL SIMULATION PROTOCOL):

ALSP resulted from DARPA effort to identify mechanisms suitable to facilitate the integration and interoperation of existing (so-called "legacy") constructive training simulations [BP97]. ALSP permits multiple warfare simulations representing distinct segments of a battlefield, to interact with each other through a common, message-based protocol interface.

ALSP Infrastructure Software (AIS) provides the infrastructure that permits multiple simulations to interact. AIS principles include the following concepts: no central node, geographic distribution of simulations, object ownership at the attribute level, a message-based protocol for the exchange of information, a conservative time synchronization approach, a common data representation, and the ability for simulations to use their existing architecture.

The AIS consists of several components. The common module provides data and time management services to each simulation. The broadcast emulator distributes messages. The control terminal provides the means for an operator to monitor and control ALSP components. Finally, the confederation management tool provides the means for an operator to monitor and control multiple AIS components [MITRE95].

1.4.2 HIGH LEVEL ARCHITECTURE (HLA)

The High Level Architecture (HLA) provides a common framework and approach for distributed simulations and virtual worlds to share information and capabilities, to expand interoperability, and to promote reuse and extensibility [D98]. HLA is not software. It is an architecture that provides standard methods of defining how distributed simulations will communicate. It is a set of specifications that define data objects. These standards are specified in the HLA Interface Specification and the Object Model Template (OMT). HLA was developed by U.S. Department of Defense (DoD) Modeling and Simulation Office (DMSO). The HLA was approved as DoD's technical architecture for modeling and simulation on 10 September 1996.

The HLA is comprised of three elements:

HLA Rules: A set of rules, which must be followed to achieve proper interaction of simulations (federates) in a federation. These describe the responsibilities of simulations and of the runtime infrastructure in HLA federations [DoD98a].

The HLA object model template, HLA interface specification, and HLA rules are currently in the formal review and balloting process for acceptance to become IEEE standard 1516. This procedure encompasses two formal review periods and two formal ballots culminating in IEEE acceptance in year 2000.

RATIONALE FOR HLA:

Following reasons lead the M&S community to develop a new framework:

- No single, monolithic simulation can satisfy the needs of the users.
- All uses of simulations and useful ways of combining them cannot be anticipated in advance
- Future technological capabilities and a variety of operating environments must be accommodated.
- Specific simulation functionality should be separated from general purpose supporting runtime infrastructure.

Before going into detail, some frequently used terminology are explained [DMSO98a]:

Federate: one simulation. A member of a federation (e.g. could represent one platform, like a ship, a cockpit simulator)

Federation: A named set of interacting simulations or federates (e.g. could represent an aggregate, like an entire national navigation simulation)

Federation Execution: A session of a federation executing together.

OBJECT MODEL TEMPLATE:

The HLA is designed to facilitate interoperability. Hence, the OMT is designed to provide a means for open information sharing across the simulation community. The OMT does not constrain the content, but provides a streamlined format for communicating to the other users, who may reuse the simulation, and the data inputs and outputs of the simulation. HLA object models are being documented using OMT components, which represent information about classes of objects, their attributes and their interactions in a tabular format using tables. The figure 1.3 summarizes the contents of OMT.

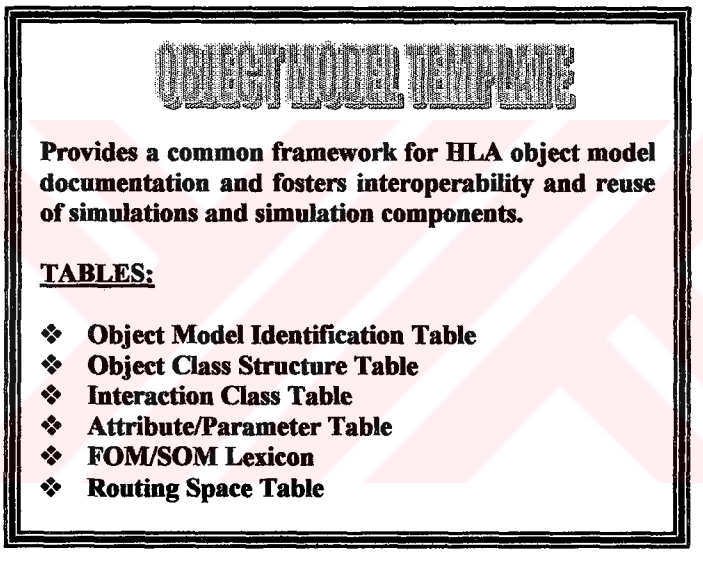


Figure 1.3 Object Model Template Contents

HLA separates data and architecture. It prescribes that OMT objects and interactions defined according to the OMT can be constructed and exchanged with no adjustments to HLA-compliant software.

The HLA specifies three types of object models: the HLA Federation Object Model (FOM), the HLA Simulation Object Model (SOM), and the HLA Management Object Model (MOM).

T.C. YÜKSEKÖĞRETİM KURULU
DOKÜMANTASYON MERKEZİ

FOM: A description of all shared information (objects, attributes, interactions, and associations) between federates essential to a particular federation.
SOM: Describes the simulation (federate) in terms of the types of objects, attributes, and interactions it can offer to potential federations.
MOM: Identifies classes and interactions related to federation management.

INTERFACE SPECIFICATION:
The interface specification identifies how federates will interact with the federation and, ultimately, with one another. The RTI provides services to federates in a way that is analogous to how a distributed operating system provides services to applications. Summary of interfaces:

Federation Management: coordinates federation-wide activities throughout the life of a federation execution. Interface functions include creation/destruction of a federation execution, joining and resigning a federate, coordination of federation save and restore, and coordination of federation synchronization points.
Declaration Management: allow federates to specify the types of data they will send or receive by object class and attribute name and by interaction class from the FOM. Interface functions include specification of publishing and subscription interests.
Object Management: supports life cycle activities of objects and interactions. The services enable the creation, modification and deletion of objects and interactions.
Ownership Management: allows federates to transfer ownership of object attributes to other participants in the simulation. The services offer both "push" and "pull" based transactions.
Time Management: provides useful services for setting, synchronizing, and modifying simulation clocks.
Data Distribution Management (DDM): allow federates to specify the distribution conditions for the specific data they send or ask to receive. RTI

uses this information to route data from producers to consumers based on DDM declarations.

HLA RULES:

The HLA Rules describe the responsibilities of federates and their relationships with the RTI. There are ten rules. Five relate to the federation and five to the federate.

Federation Rules:

1. Federations shall have an HLA Federation Object Model (FOM), documented in accordance with the HLA Object Model Template (OMT).

2. In a federation, all representation of objects in the FOM shall be in the federates, not in the run-time infrastructure (RTI).

3. During a federation execution, all exchange of FOM data among federates shall occur via the RTI.

4. During a federation execution, federates shall interact with the run-time infrastructure (RTI) in accordance with the HLA interface specification.

5. During a federation execution, an attribute of an instance of an object shall be owned by only one federate at any given time.

Federate Rules:

6. Federates shall have an HLA Simulation Object Model (SOM), documented in accordance with the HLA Object Model Template (OMT).

7. Federates shall be able to update and/or reflect any attributes of objects in their SOM and send and/or receive SOM object interactions externally, as specified in their SOM.

8. Federates shall be able to transfer and/or accept ownership of attribute dynamically during a federation execution, as specified in their SOM.

9. Federates shall be able to vary the conditions (e.g., thresholds) under which they provide updates of attributes of objects, as specified in their SOM.

10. Federates shall be able to manage local time in a way which will allow them to coordinate data exchange with other members of a federation.

BRIEF COMPARISON OF DIS PROTOCOLS:

Today, there exist three examples of standardized domain of interests in terms of protocols. Those are 1) SIMulator NETworking (SIMNET), 2) DIS protocol-Protocol Data Units for Distributed Interactive Simulation (PDU for DIS) - IEEE 1278.1a-1998, and 3) the Aggregate Level Simulation Protocol (ALSP). While the first two of those have a common core interests consisting of classical, physical interactions between real world objects (entity based), the ALSP's interest primarily aggregates entities. Therefore, the ALSP and the nature of the corresponding simulations (statistical considerations, closed loop, time stepped, even driven) do not have much in common with the former protocols and the corresponding virtual simulators. The process of aggregation requires a high level description of interacting objects in contrast to the entity resolution case. From this, a priori interoperability follows for PDU for DIS-systems in contrast to the interaction concept of ALSP based on Interface Control Documents (ICD).

The High Level Architecture (HLA) aims to be the smallest common denominator of descriptive elements in order to finally cover a maximum domain of interests. That's why HLA is a high-level system-modeling concept rather than a protocol suite. Definitely, HLA promotes simulations (federates) of different nature (live, virtual, constructive) to become interoperable based on a common Object Model Template (OMT). This OMT comes along with an appropriate Data Interchange Format (DIF) which one has to agree upon in order to exchange compatible information among federates. To this end, the OMT DIF announces classes of objects and interactions without using the implications of the object oriented paradigm such as abstract classes and polymorphism. However, reusability is gained with the help of inheritance between classes within the OMT.

A comparison of HLA with the IEE DIS protocol:

- Existing DIS protocol applies to only real-time, platform level niche of M&S market. HLA applies to multiple time management schemes.

- Embedding data in architecture has caused the DIS protocol to be inflexible and ineffective. HLA separates data from architecture; data evolve as required by applications.
- DIS protocol uses full broadcast distribution approach. Data should be selectively passed among simulations in order to scale the simulation from a network viewpoint. HLA selectively passes data among simulations.
- DIS does not possess all the simulation services required by simulations. HLA does.
- DIS uses message-passing mechanism for data exchange, while HLA uses object-exchange mechanism.

A comparison of HLA and existing ALSP Protocol:

- ALSP applies to discrete-event, logical-time niche of M&S market. HLA applies to multiple time management schemes.
- ALSP is designed to accommodate legacy simulations. HLA is new, more robust approach designed in from onset.

1.4.3 UNIFIED MODELING LANGUAGE (UML)

The UML is an industry standard language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system [OMG99]. It can be used with all processes, throughout the development life cycle, and across different implementation technologies. The UML combines the best from *Data Modeling Concepts, Business Modeling (workflow), Object Modeling, and Component Modeling.*

All analysis and design of NSTMSS federates is based on UML version 1.1.

The UML notation is a fusion of the notations of Booch, OMT (Object Modeling technique), OOSE (Object Oriented Software Engineering) and others. UML is straightforward, homogenous, and consistent. Awkward, redundant and superfluous symbols have been eliminated, in order to favor a better visual rendering [MM97].

There are many views of a house under construction-the floor plan, the electrical circuit layouts, and the plumbing layouts, to name a few. There are also views of a software project under development. Each view represents a different aspect of the visual model. UML defines nine types of diagrams to represent the various modeling viewpoints. A diagram provides the user with the means of visualizing and manipulating model elements. Here is a list of the diagrams:

- ***Use Case Diagrams*** represent the functions of the system from the user's point of view.
- ***Sequence Diagrams*** are the temporal representation of objects and their interactions.
- ***Collaboration Diagrams*** are spatial representations of objects, links, and interactions.
- ***Class Diagrams*** represent the static structure in terms of classes and relationships.
- ***Object Diagrams*** represent objects and their relationships
- ***Deployment Diagrams*** represent the deployment of components on particular pieces of hardware.
- ***Activity Diagrams*** represent the behavior of an operation as a set of actions.
- ***Statechart Diagrams*** represent the behavior of a class in terms of states.

Sequence and collaboration diagrams can be grouped together under the more general title of *interaction diagrams* [MM97].

The UML enables and promotes (but does not require or mandate) a use-case-driven, architecture centric, iterative and incremental process that is object-oriented and component based. Use cases are used to manage and provide focus for a problem-solving effort; architecture is used to manage complexity and maintain integrity and focus as a solution to a problem;

iterations and increments are used to repeatedly apply a process to evolve a solution to a problem.

In this project, the UML is heavily used to:

- Display the boundary of the system & its major functions using use cases and actors,
- Illustrate use case realizations with interaction diagrams,
- Represent a static structure of a system using class diagrams,
- Reveal the physical implementation architecture with component & deployment diagrams,

1.5 TOOLS

1.5.1 PERFORMER

DESCRIPTION

IRIS Performer provides a programming interface (with ANSI C and C++ bindings) for creating real-time graphics applications and offers high-performance rendering in an easy-to-use 3D graphics toolkit [SGI97]. IRIS Performer interfaces to both the OpenGL® graphics library and the IRIS Graphics Library™ (also known as IRIS GL™); these libraries, combined with the IRIX™ operating system, form the foundation of a suite of tools and features for creating real-time 3D graphics applications on Silicon Graphics ® systems.

FEATURES

A visual database, also known as a *scene*, contains state information and geometry. A scene is organized into a hierarchical structure, specifically as a directed-acyclic graph. The scene hierarchy supplies definitions of how items in the database relate to one another. Additionally logical and spatial organization information of the database is also held within the same hierarchy. The scene hierarchy is traversed by visiting the nodes in depth-

first order and operating on them. IRIS Performer implements several types of database traversals, including application, cull, delete, draw, and intersect.

A logical hierarchy consists of a set of objects grouped under a common parent according to the object type regardless of position in the database whereas with a spatial hierarchy, the parent nodes that objects are grouped under correspond to a particular position in the scene.

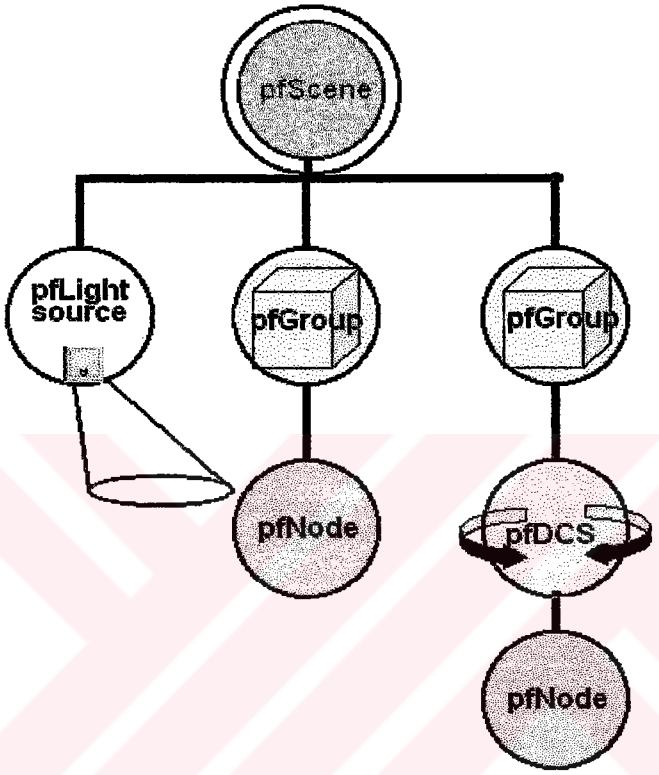


Figure 1.4 Database Hierarchy Example of a Performer Scene

The scenes in NSTMSS processes are logical hierarchies. Geometry for the visual database may be created utilizing Performer's high speed rendering library (*libpr*). A scene is rendered into a *channel* or several channels, which are culled and drawn in the hardware graphics pipeline. A view into the world is described by a channel. The remainder of the nodes in the database includes moving and non-moving coordinate systems, nodes containing geometry, animations, and nodes to switch between different sets of geometry. The actual geometry of the scene, including light-points, is located at the leaf nodes.

IRIS Performer consists of two main libraries, *libpf* and *libpr*, and four associated libraries, *libpfdu*, *libpfdb*, *libpfui*, and *libpfutil*. *Libpf* is the main library, which handles multi-processed database traversal and rendering and also contains *libpr*, which performs the optimized rendering, state control, and other functions fundamental to real-time graphics. *Libpfdu* is the library of scene geometry building tools, which greatly facilitates the construction of database loaders and converters. *Libpfutil* and *Libpfui* are utility functions and user interfaces respectively [SGI97].

1.5.2 RUN-TIME INFRASTRUCTURE (RTI)

Run-Time Infrastructure (RTI) provides a specification of the functional interfaces between federates and the RTI. Interfaces are divided into six service groups, which are summarized in the preceding "HLA-Interface Specification" section 1.4.2. Each service specification includes: name and descriptive text, supplied arguments, returned arguments, pre-conditions, post-conditions, exceptions, and related services. RTI Application Programmer Interfaces (APIs) are in CORBA IDL, C++, Ada95, and Java. We used C++ API in NSTMSS federates.

MAJOR COMPONENTS:

RTI software, freely available from DMSO, is currently comprised of the *RTI Executive process (RtiExec)*, the *Federation Executive process (FedExec)* and the *libRTI library* [DMSO98b]. As illustrated in Figure 1.5, each executable containing federates incorporates libRTI. Federates may exist as independent processes or be grouped into one or more processes. A federate may participate in more than one federation. RTI software can be executed on a standalone workstation or executed over an arbitrarily complex network.

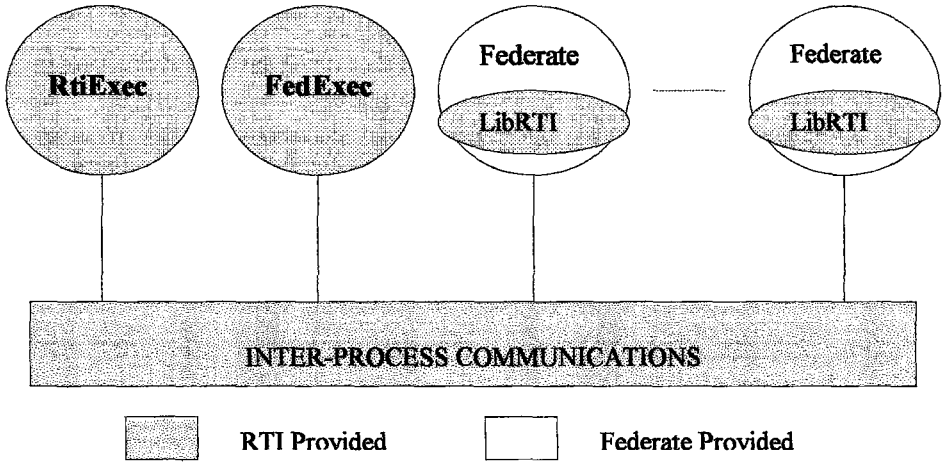


Figure 1.5 RTI Components At-a-Glance

RTIEXEC: *RtiExec* (RTI Executive) is a global process. Each federate communicates with RtiExec to initialize RTI components. The RtiExec's primary purpose is to manage the creation and destruction of FedExecs. The RtiExec directs joining federates to the appropriate federation execution. RtiExec also provides a naming service for federation executives and ensures that each FedExec has a unique name.

FEDEXEC: Each *FedExec* (Federation executive) manages a Federation. It allows federates to join and to resign, and facilitates data exchange between participating federates. A FedExec process is created by the first federate to successfully invoke the "Create Federation Execution" service for a given Federation Execution name. FedExec assigns a unique handle to each federate joining the federation.

LIBRTI: The C++ library, *libRTI*, provides the services specified in the *HLA Interface Specification* to federate developers. Federates use libRTI which communicates with the RtiExec, a FedExec, and other federates to invoke HLA services.

MANAGEMENT AREAS:

RTI partitions the exchanges that take place between federate and Federation into six management areas of the FedExec life cycle, as shown in Figure 1.6.

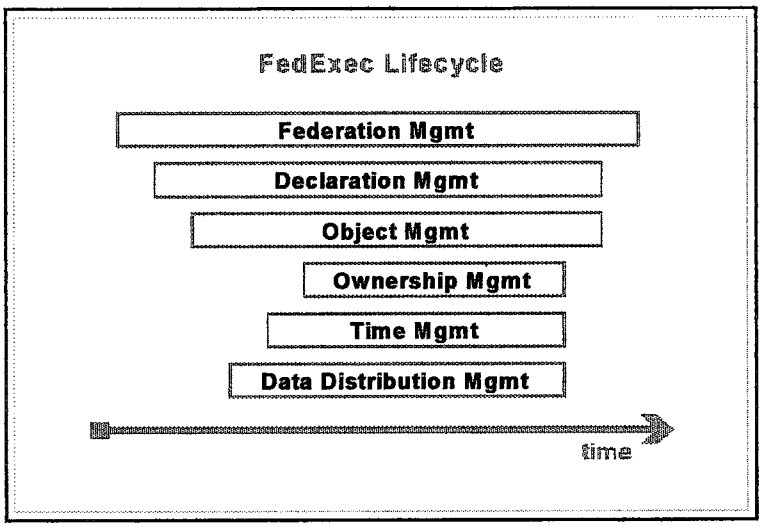


Figure 1.6 Federation Execution Lifecycle

Federation execution lifecycle starts with the federation creation and participation to it (Federation Mgmt.). After joining, the federate declares its interests to federation (Declaration Mgmt.) and creates its objects and interactions (Object Mgmt). Later, the federate updates its object according to its specified data space (Data Distribution Mgmt.) and time management scheme (Time Mgmt.). Federate can pass the ownership of its created object to another federate (Ownership Mgmt.). After completion of its execution, the federate deletes the created object, erases its interests, resigns from federation, and tries to delete the federation execution; if it is the last federate, it succeeds, and the lifecycle ends.

1.5.3 RATIONAL ROSE

Rational Rose is one of the visual modeling Computer-Aided Software Engineering (CASE) Tool based on Unified Modeling Language (UML) [RSC98]. CASE provides the engineer with the ability to automate manual activities and to improve engineering insight. Rose lets you easily document,

T.C. YÜKSEKÖĞRETİM KURULU
DOKÜMANTASYON MERKEZİ

represent, and generate interface codes for software architectures. In addition, Rose serves all the benefits of visual modeling including:

- ❖ Shorter development cycles through support for controlled iterative development.
- ❖ Improved software quality through user-focused development.
- ❖ Support for large projects and multi-project organization through a scalable team environment.
- ❖ Focus for software architecture and components.
- ❖ Integration with legacy systems through Reverse Engineering capabilities and extensibility interfaces.

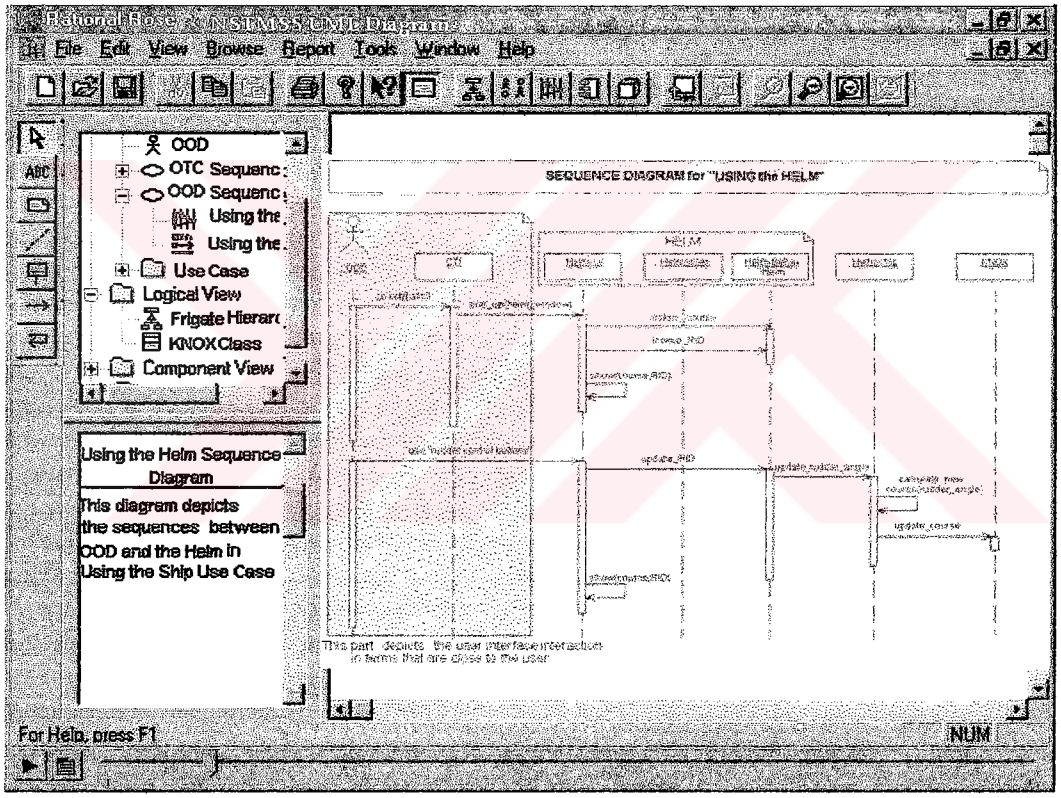


Figure 1.7 Rose User Interface

In this project, Rose is heavily used for UML diagram documentation in analysis and design. Rose supports the UML use-case diagrams, sequence diagrams, collaboration diagrams, class diagrams, state-transition diagrams, component diagrams, and deployment diagrams. Rose provides a GUI for

constructing all those diagrams. The Rose GUI is depicted in the figure 1.7. Rose is used for analysis and design documentation. As well Code samples (class interfaces) were also generated automatically by Rose from UML models. Although the generated interface code was poor, it helped us to understand the mapping of UML diagrams into C++.

1.5.4 OBJECT MODEL DEVELOPMENT TOOL (OMDT)

The primary purpose of the Object Model Development Tool (OMDT) is to automate the process of constructing Simulation Object Models (SOMs) and Federation Object Models (FOMs) [AEGIS98]. These object models are key components in developing HLA federations. The OMDT provides the user with an interface to object models consistent with the tabular views defined in the HLA OMT specification. As object models are constructed, the OMDT performs consistency checking to ensure that the specification complies with the rules defined by the HLA OMT.

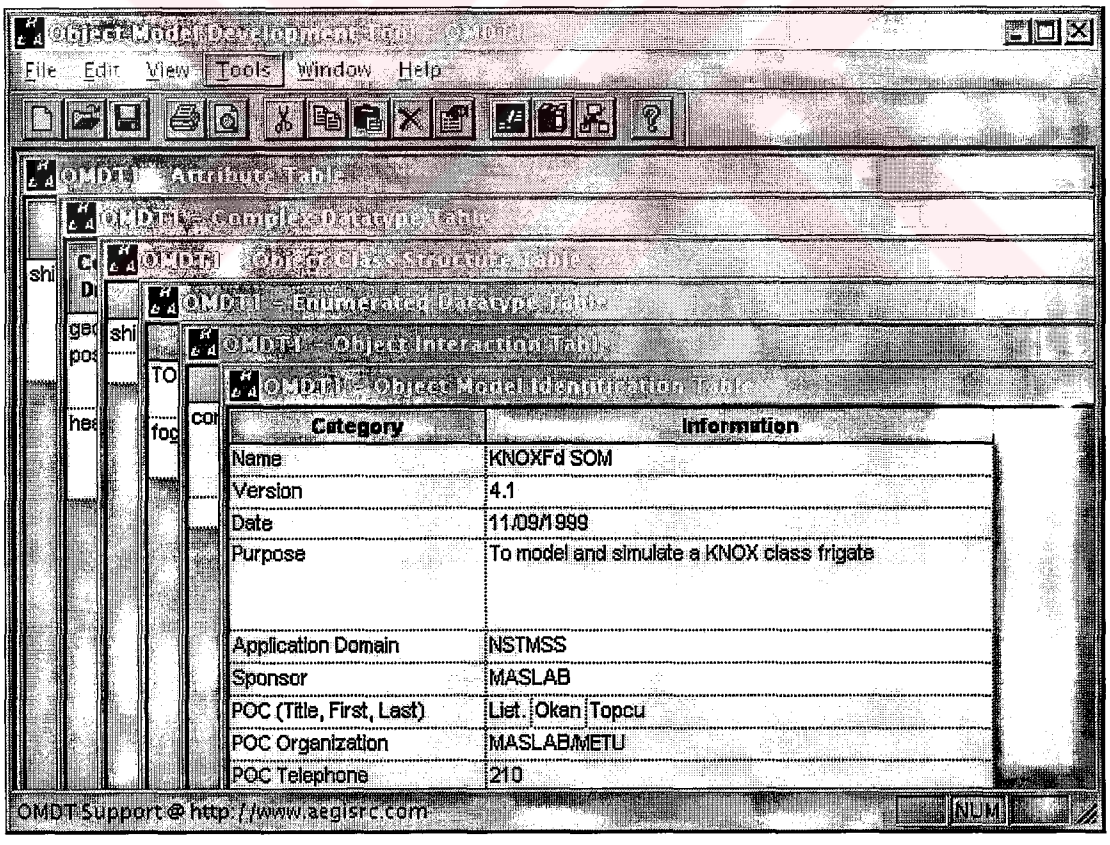


Figure1.8 OMDT User Interface

In addition to automating the construction of object models, the OMDT provides the user with an interface to the DMSO *Modeling and Simulation Resource Repository (MSRR)*[2]. This interface allows users to browse, download and upload object models to the MSRR.

Once an object model has been constructed, the tool can generate the Federation Execution Details (FED) file required by the Runtime Infrastructure (RTI) to execute the federation. The OMDT supports the specification of additional information not defined by the HLA OMT, but required for full FED generation.

1.6 RELATED WORK

1.6.1 NPSNET

NPSNET (Naval Postgraduate School Networked Environment) is a large scale, distributed, virtual environment developed at the Naval Postgraduate School, starting in 1990. It is used as a testbed for Distributed Interactive Simulation Protocol for DoD simulations, and for incorporating new input devices (e.g. Head Mounted Displays (HMD), Flight Control Sticks (FCS), etc.) important in virtual environment. NPSNET is implemented in C/C++ and in the Performer Graphics API. Currently, NPSNET is in its fourth major version (NPSNET-IV). It includes a suite of complementary software applications such as network management tools and various interfaces. NPSNET-IV uses DIS protocol v2.0.3 for network communications [ZPFB93, ZMPB94, Z93].

1.6.2 FULL MISSION SHIPHANDLING BRIDGE SIMULATORS

The full-mission shiphandling simulator's bridge has been constructed to replicate a real ship's bridge and equipped with the safety and navigation equipment normally found on a real ship's bridge such as radar repeaters,

[2] http://www.msrr.dmso.mil

navigation equipment, communications equipment, course and speed indicators, and engine status displays. The simulation software allows for complex or simple training exercises and scenarios that cover major shiphandling techniques. Advanced hydrodynamics and visual imagery enhances training effectiveness. Windows are provided to look out at the virtual waterway, which is projected on large screen displays located behind the windows. To maneuver the simulated ship, the officer utters standard voice rudder and engine commands to another person also located on the bridge at a control console or helm. There is one full-mission shiphandling bridge simulator training facility located in Surface Warfare Training Center in Turkish Navy [SA99]. Besides its high fidelity, it is a stand-alone, expensive, and non-portable simulator.

The full-mission simulators are very expensive and require large support organizations, and often specialize in specific domains (e.g. shiphandling). Therefore, interfacing the existing training simulations increase the utility of individual full-mission simulators.

1.7 SUMMARY OF CHAPTERS / THESIS OUTLINE

The preceding sections of this chapter outline the background, motivation and objectives as well as general problem definition. The remaining chapters are broken down as follows:

- Chapter 2 presents an overview of the whole NSTMSS architecture design philosophy and time management, also includes initialization and termination of the environment.

- Chapter 3 is dedicated to simulation processes: the NSTMSS federates. This chapter includes the design and implementation of KNOX Federate and its user interface, and the tactical level simulation, OTC federate

structure and its user interface. It also presents an overview of federation management processes.

- Chapter 4 describes experimental hardware architecture and network issues. This chapter includes the Federation Execution Planner's Workbook for NSTMSS.

- Chapter 5 presents development methodology of NSTMSS and discusses the lessons learned in FEDEP and UML.

- Chapter 6 outlines conclusions reached as a result of this research as well as recommendations for future research in this area.

- Appendices A through C give all the analysis and design documentation of NSTMSS based on UML.

- Appendix D gives the implementation specific details and explains the algorithms used in development of federates.

- Appendix E presents the practical matters, which include RTI, FEDEP, FED file, and C++ issues.

- Appendix F covers a new notation for federation design developed for the consistency between NSTMSS documentation.

- Appendix G includes HLA Federation Object Model (FOM) and Simulation Object Model (SOM) documentation for reuse.

- Appendix H presents a glossary of naval terminology used in NSTMSS.

CHAPTER 2

NSTMSS ARCHITECTURE

It is important to understand the overall architecture of NSTMSS, before going in detail of federation components. This chapter examines the system at top level and it reveals the system components, federation objects and interactions, and the way the components interact.

2.1 SYSTEM COMPONENTS

Distributed among machines on a network, simulations achieve greater utility. A single machine can be useful as a vehicle trainer or as a way for an operator to engage computer-driven entities. But a broader goal is to allow human interaction, to engage others through individual control of entities, or to develop command-and-control structures within teams. The overall intention is to bring heterogeneous federates from different simulation domains together to form a distributed naval simulation within a virtual environment and to coordinate all these activities in space and time.

Major components of NSTMSS are shown in Figure 2.1. Simulation management facilities manage and monitor the whole simulation and collect the critical performance data. OTC is a tactical level simulation and provides interfaces for tacticians to handle the whole STG. In the figure, each black point represents a ship, which is in fact controlled by a ship-handling simulator in the federation. Shiphandling simulator is a platform level simulation. It provides interfaces for the OOD to maneuver the ship and

renders a 3D view of virtual environment and other entities in the federation. Many ships can participate in the virtual environment.

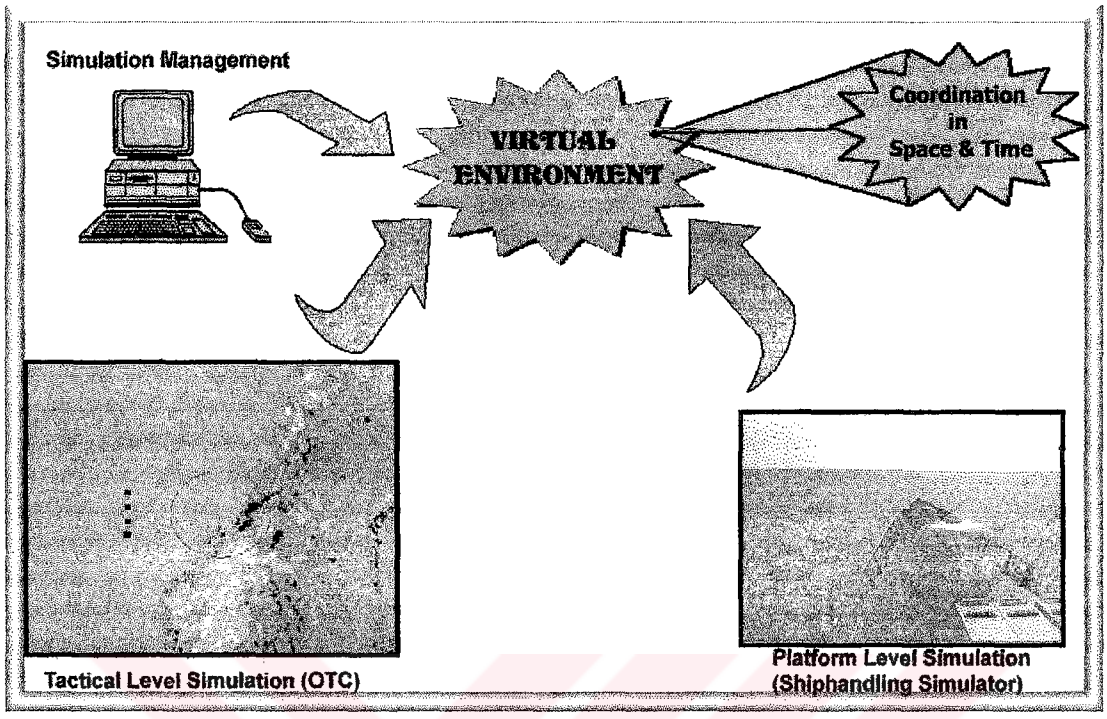


Figure 2.1 NSTMSS Components

2.2 DESIGN PHILOSOPHY

In a distributed interactive simulation system, the analysis and design of simulation entities (federates) is not the only issue. The design of virtual environment (federation), which is being constructed by these federates, should be considered seriously.

Today, it is recognized that designing a distributed software system starts at the architectural level; in HLA domain, this is called federation design. The purpose at this stage is to define a set of software components, the way they interact, and the way they are mapped to an execution platform (OS, middleware, etc.). Decisions taken at the architectural level have an important and long lasting impact on the system.

All the analysis and design at the component level (federate level) are performed by using iterative and incremental object oriented analysis and design methodology, and are being documented and discussed widely in appendices A, B, and C using UML notation.

All federates (e.g. KNOXFd, OTCFd...) are separate processes in functionality. The aim of the federation design is to bring these separate simulations together to form a common environment which functions as a whole. A new federate can easily be adapted to NSTMSS, or an existing federate can easily be reused in another federation, because NSTMSS architecture defines the rules of the game, the static architectural view, the common object and interaction classes that federates use, Federation Object Model (FOM), and the capabilities of the existing federates.

2.3 FEDERATION DESIGN

Software components are classified into four groups:

- RTI-Provided Processes
- Simulation Entity Processes/Federates
- Environment Generation Processes/Federates
- Federation Management Processes

Simulation Entity Group consists of the real life entities simulated in virtual environment. There are two different kinds of federates in Simulation Entity Group. The ships are brought into federation by the KNOX Federates (KNOXFd's) which are platform level simulations allowing a person to steer the ship in real-time. The other federate is OTC Federate (OTCFd) which is a tactical level simulation allowing a user to control the surface task group, which is composed of the ships in the virtual environment.

Environment Federate, according to scenario,

- Describes the scenario area and its limits,
- Creates some environmental effects such as fog, dynamically through the federation,
- Prescribes the time of day through the federation, and provides command line interface to instructor switching the value (e.g. from DAY to EVENING).

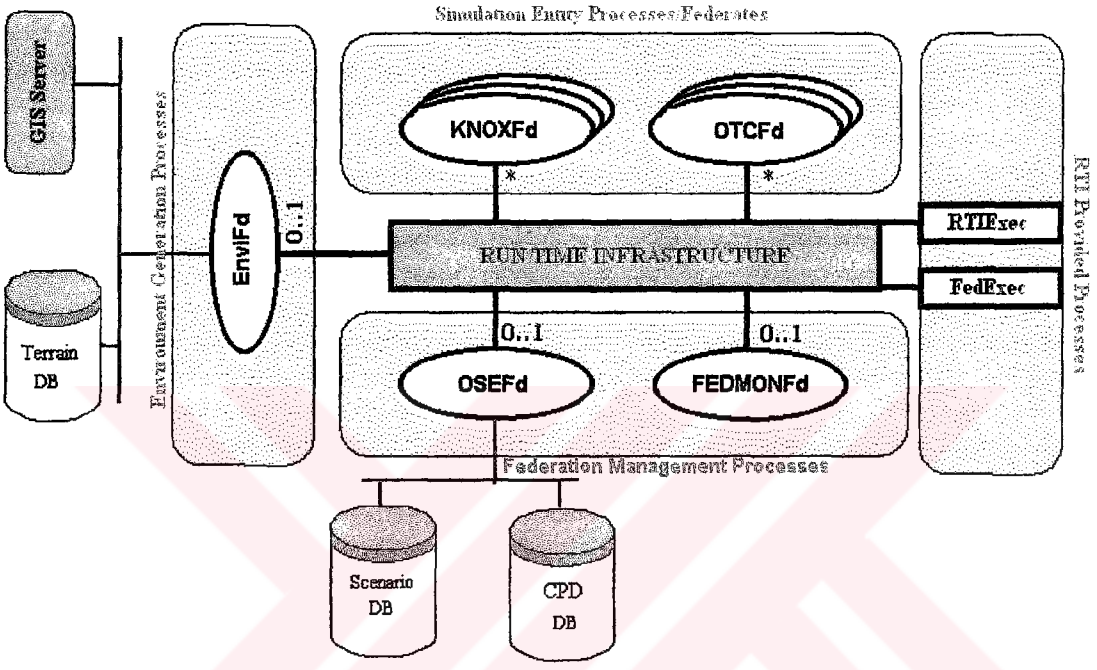


Figure 2.2 Architectural Overview of NSTMSS

Figure 2.2 graphically depicts the software components and their processes of NSTMSS (please refer to appendix-D for an overview of the notation). Multiplicity information on the figure explicitly expresses that while many KNOX federates and OTC federates can participate in the environment, at most one OSE, Federation Monitor and environment federate can participate.

2.3.1 RTI PROVIDED PROCESSES

RTI Provided Processes handle inter-object communication between processes [DMSO98b].

RtiExec (RTI Executive) is a global process. The RtiExec's primary purpose is to manage the creation and destruction of federations. Each federate communicates with RtiExec to initialize RTI components. The RtiExec directs joining federates to the appropriate federation execution. RtiExec also provides a naming service for federation executives and ensures that each federation has a unique name.

Each *FedExec (Federation executive)* manages a federation. It allows federates to join and to resign, and facilitates data exchange between participating federates. A FedExec process is forked by the first federate to successfully invoke the "Create Federation Execution" service for a given Federation Execution name. FedExec assigns a unique handle to each federate joining the federation.

2.3.2 PUBLISH/SUBSCRIBE (P/S) PATTERN

This pattern defines an indirect form of communication where software components use a third party, called a *broker*, in order to exchange data. At run time, software components can specify to the broker a set of data types they can provide, and a set of data types they are ready to receive according to Federation Object Model (FOM) and Federation Execution Data (FED). The broker is an independent computing module, namely Run Time Infrastructure Executive (RtiExec).

Publishing means providing data. Object classes and attributes, and interaction classes that the federate is able to update or send are published.

Subscribing means expressing interest in receiving certain data. Object classes and attributes, and interaction classes that the federate is interested to receive are subject to subscribing.

The entire notation is constructed for the sake of compatibility of NSTMSS documentation and is described in detail in appendix F. This notation fills a gap of graphical representation of federation design. Developed notation also satisfies my one of the secondary objectives: FEDEP methodology improvement.

As a design principle, for the sake of bandwidth utilization, a federation object or interaction is only sent through network when interested federates join into environment, instead of periodically sending a heartbeat-like message. For example, when EnviFd joins the federation, it creates environment object and it looks for whether an interested federate (a federate which has subscribed to environment object) exists or not. If it exists, it sends the environment object attributes and then goes to sleep until another interested federate joins the environment, resulting diminished network load.

GENERAL P/S DIAGRAM:

General P/S diagram of NSTMSS federation depicts all the capability of existing federates in terms of objects and interactions that they produce or they are interested in. General P/S diagram in the Figure 2.3 depicts the system in a static view, which means that all the federate interests are being hard-coded at system start-up. At run-time, the interests of federates can be changed.

The rectangular shapes in the figure represent the NSTMSS objects, which are exchanged through federation. The oval shapes represent the NSTMSS federates. The line from a federate to an object means that the federate has the capability to publish the object, on the other hand a line from an object to a federate means that the federate subscribes to object or is interested with published object and its attributes (see appendix-F for further explanation).

HLA allows *Routing Space* definitions to define the object-exchange regions by utilizing RTI data distribution management capabilities. All the exchanges of objects take place in *Default Space* predefined by RTI except that the communication interaction exchanges take place in *Tactical Communication Space*, which widely expressed in chapter 3.

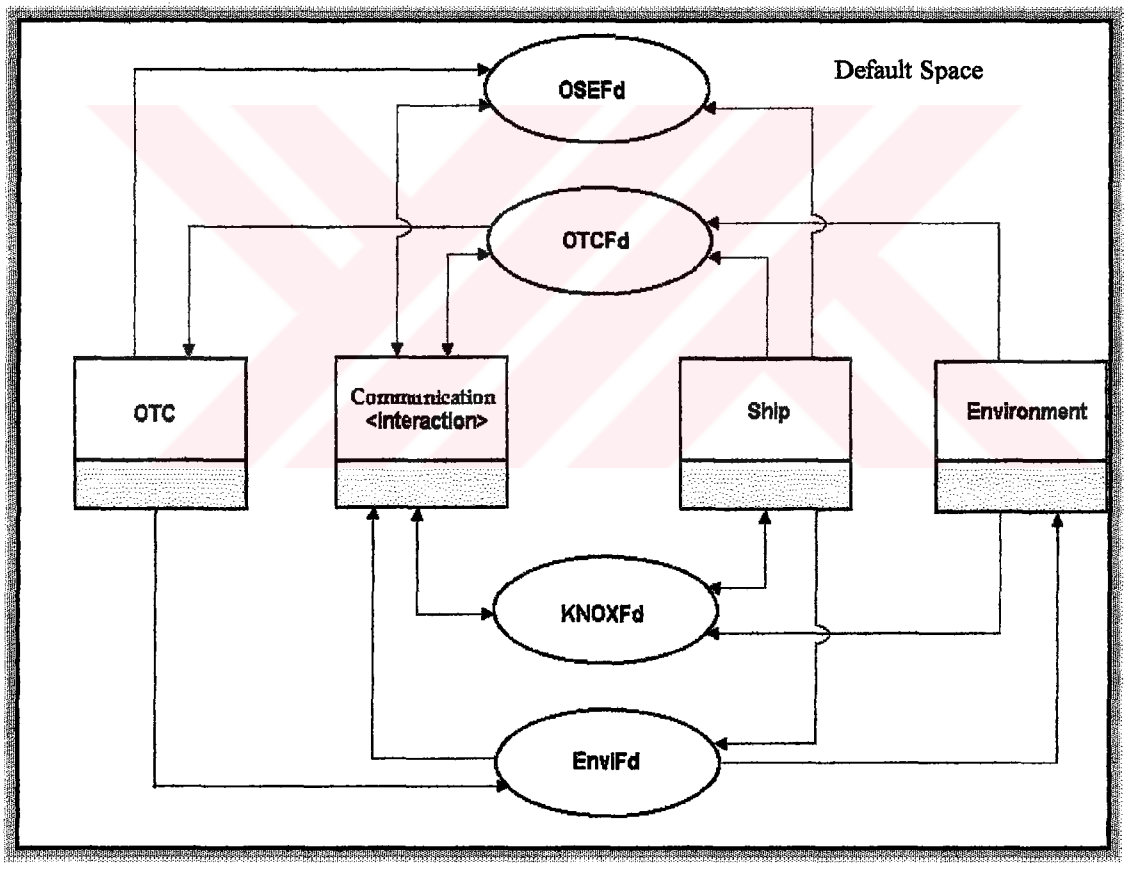


Figure2.3 General P/S diagram of NSTMSS federation

Ship object class represents any ship in real life. About ship objects, the figure expresses that only Knox federates (KNOXFd) are responsible to create and update the ship object. When an instance of ship object is created or updated, the other Knox, OSE, and OTC federates are informed, and they take care of it (e.g. they add the newly-discovered ship to their Scene-Graphs to render it).

Communication is an interaction class. Interaction classes describe types of events. The primary difference between objects and interactions is persistence: Objects persist, interactions do not. OSEFd, KNOXFd and OTCFd may receive messages (or send messages) as events.

CLASS-BASED P/S DIAGRAMS:

Class-based P/S diagrams show all the common NSTMSS objects and interactions, and their parameters and attributes one by one while depicting the interests of federates over these objects and interactions.

The detailed information about object classes, interaction classes, their attributes and parameters is given in appendix G (Object Model Template for NSTMSS).

Environment object-class P/S diagram: Figure 2.4 depicts the environment object class and its attributes. Environment object encapsulates the file name and the limits of the area, which scenario takes place. *LLC* (Lower Left Corner) and *URC* (Upper Right Corner) attributes hold the limits of the area in terms of geodetic position (refer to section 2.4 in this chapter for the explanation of the geodetic coordinate system). Environment object also controls some environmental effects such as fog and time of day (TOD). *FogStatus* attribute holds the present fog status in the common environment for the federates, while *TOD* attribute holds the present time of day.

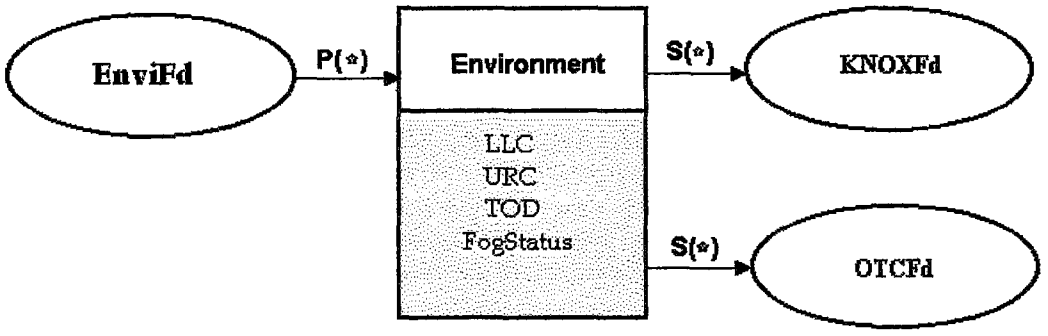


Figure 2.4 Environment Object P/S Diagram

As seen from the Figure 2.4, only the EnviFd can create and update the environment object and its attributes. KNOXFd and OTCFd are informed when an environment object is being created or updated to reflect the changes in their 3D scene in runtime.

OTC object-class P/S diagram: OTC object holds the information about Officers in Tactical Command in the virtual environment. When OTCFd joins into virtual environment (federation), it must create the OTC object and it must update its attributes. OTCFd creates the OTC object and is responsible to inform the name and the rank of who is intended to simulate. OSEFd reads the OTC object to discover the present and newly joined OTCs in the federation. EnviFd is interested with OTC object because when an OTC federate is joined into federation, the EnviFd informs the OTCFd about the present situation of the environment.

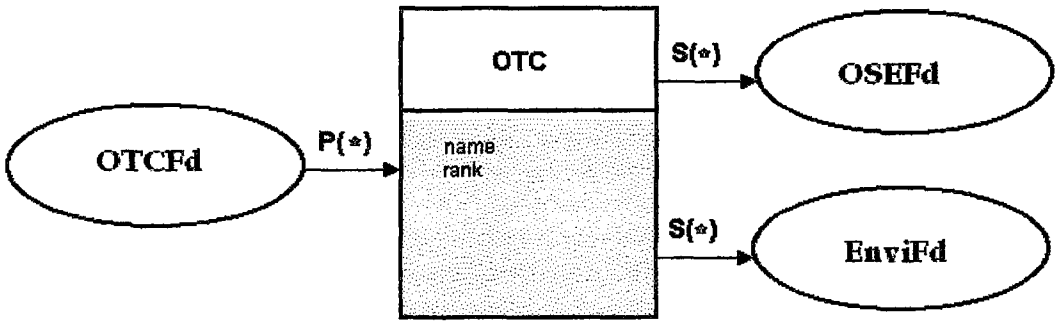


Figure 2.5 OTC Object P/S Diagram

Ship object class P/S diagram: Ship object holds the entity information (name, class, position, etc.) of the ships in the environment. *Class* attribute holds the class information of the simulated ship (e.g. KNOX, MEKO). *Name* attribute holds the name of the simulated ship (e.g. TCG MUAVENET). *Position, velocity, and heading* attributes hold the posture of the simulated ship. After KNOXFd joins the federation, it must update the ship object with its attributes. Thus, all other federates (OTCFd, OSEFd, and KNOXFd) in the federation will discover that a new KNOX frigate is joined into environment. OSEFd is only interested with class, name and position attributes to use this information in role casting. EnviFd does not track the ships in the environment, but EnviFd is subscribed to ship object because when a KNOX federate is joined into federation, it sees that a new ship just joined into environment and then informs it about the present situation of the environment.

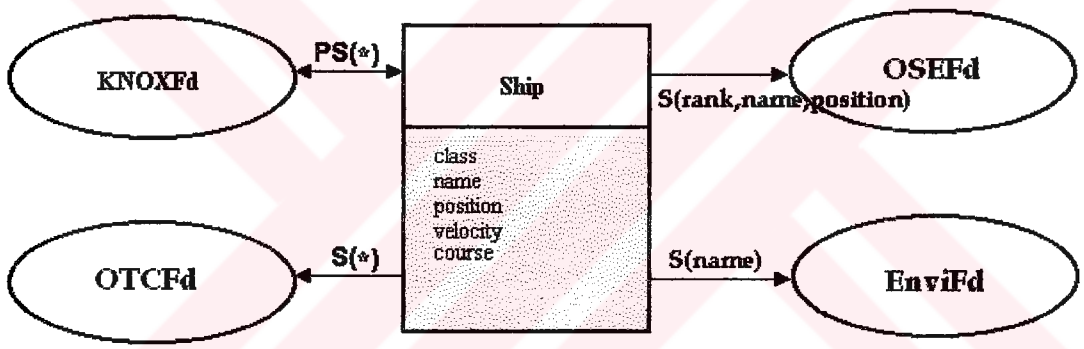


Figure 2.6 Ship Object P/S Diagram

Communication interaction-class P/S diagram: Communication interaction consists of three parameters: subject, header, and body and three sub-classes: TaskOrd, WeatherRep, and FormMsg. Each interaction is constructed, sent, and forgotten. The recipients receive, decode, and apply the interaction. Subject parameter holds the subject information. Header parameter consists of from, to, and cc information, which prescribes the sender and the receivers. Body parameter encapsulates the main message. The Figure 2.7 shows the communication interaction-class P/S diagram.

Although the parameters of three subclasses are the same, the reason for the inheritance of three subclasses is that the interest of federates is different according to the communication message type as depicted in the Figure 2.7. For example, the OSEFd is the only federate capable of sending a tasking order (TaskOrd) message, while others are only able to receive it.

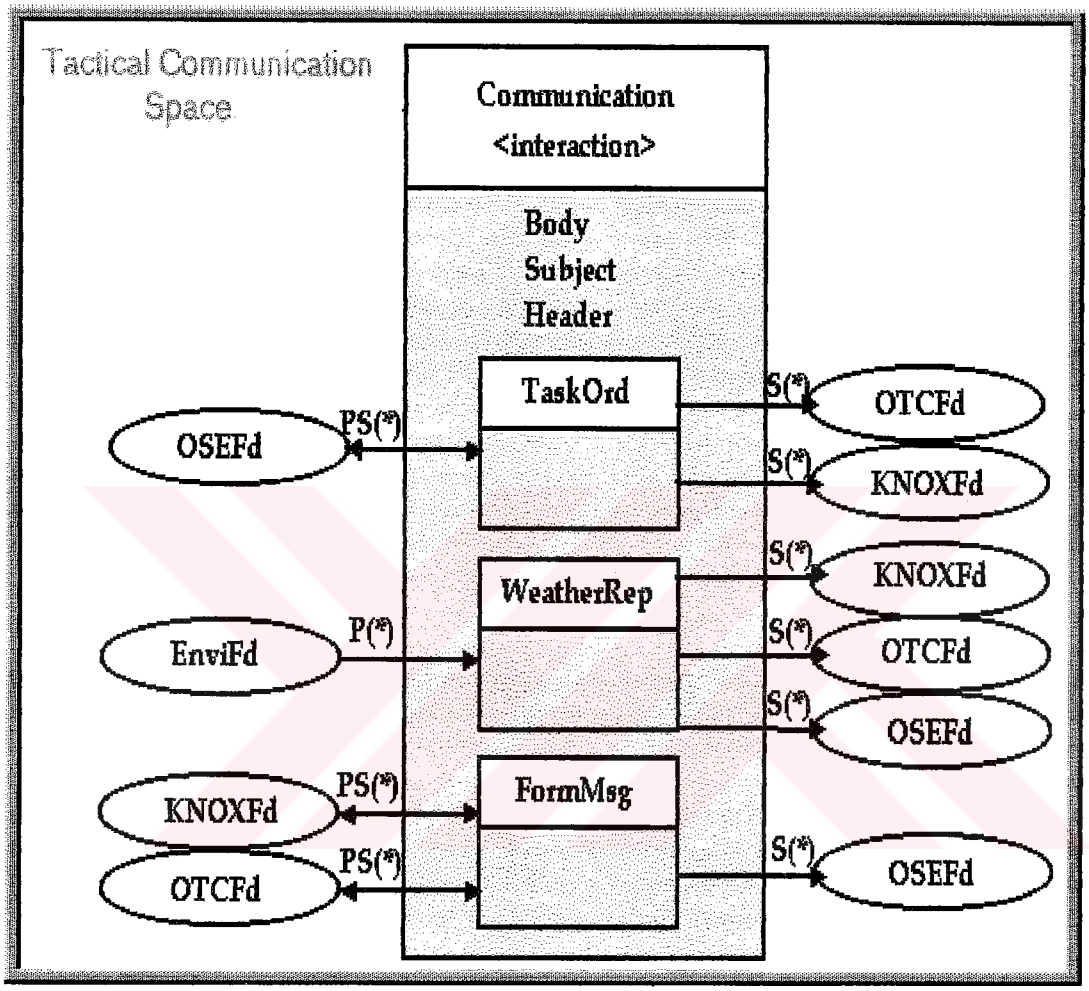


Figure 2.7 Communication Interaction P/S Diagram

2.4 INITIALIZATION AND SHUT-DOWN PRINCIPLES

A typical federation lifecycle is depicted in the figure 2.8.

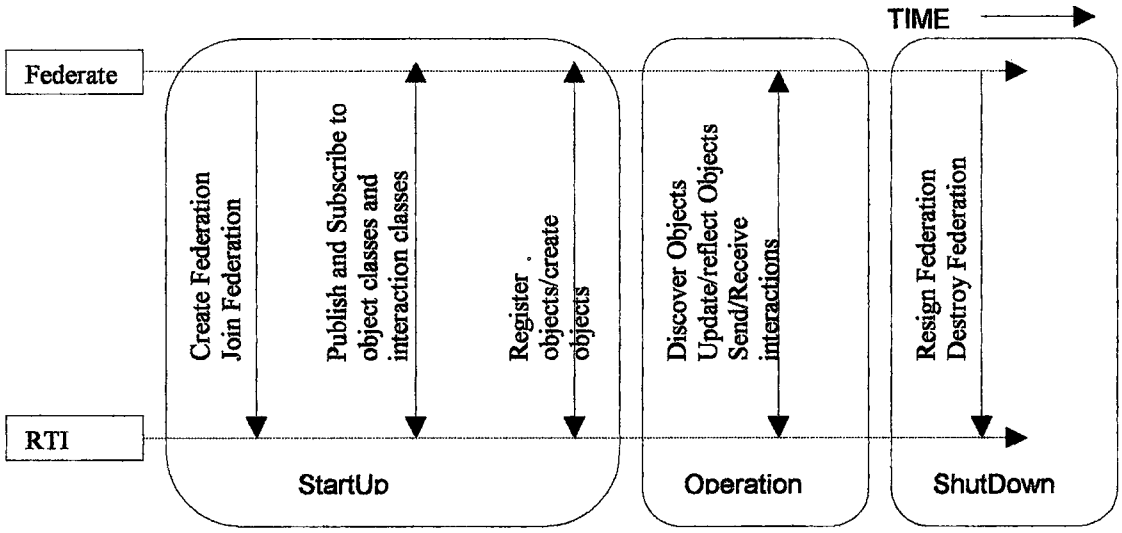


Figure 2.8 Typical Federation Execution Lifecycle

Federation execution lifecycle starts with the creation of federation, then:

- Federate joins the federation,
- Federate should inform the RTI about its capabilities and interests by publishing and subscribing the object-classes and interaction-classes,
- Federate creates objects that it will provide to other federates,
- Federate may update objects, may discover new objects created by other federates, may receive updates for subscribed objects, may send and receive interactions,
- Federate deletes objects that created by itself before leaving,
- Federate resigns

This typical federation execution lifecycle affect the design of federates. With no surprise, basic program flow of federates is divided into three phases: system initialization, main application loop, and system termination.

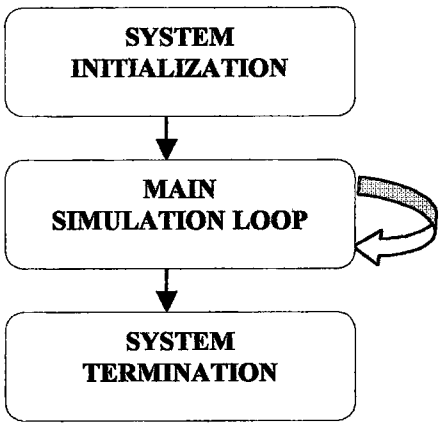


Figure 2.9 Basic Program Flow of a Typical Federate

All phases are deeply examined in chapter 3. But system initialization and termination phases include RTI initialization and termination phases, which involve federation wide principles, and therefore they concern this chapter.

The creation of the Federation is not a privilege of a particular federate to achieve more flexibility and non-centralization. Hence, the first job of any one of the NSTMSS federates is to try to create a federation. The first federate succeeds to create the federation if the specified federation does not exist; the *rtiexec* process creates a new *FedExec* process and associates it with supplied federation name, while subsequent federates receive an exception, which indicates that the federation does already exist and they directly join the federation instead of creating it. The following pseudo-code snippet

```
Create Federation
{
  try
    {
          create federation execution (federation name)
    }
  catch (federation execution already exists)
    {
      loop
          join federation (federation name, federate name)
      until (join is successful) or (# of joins attempts is exhausted)
    }
  catch (another error)
    {
          exit
    }
};
```

explain more than words; every federate shares the same algorithm for initialization of federation:

If joining the federation is attempted too quickly following the creation of federation, the FedExec process may not yet be initialized to communicate with the federate that successfully created FedExec (typically the first federate). Beforehand we can not assume which federate is the first, so the join logic will loop until the join is successful or until some predetermined number of join attempts is exhausted.

The creation of a federation requires a federation name. It designates a unique federation and the participating federates use it to join into the specified federation. All federates should agree on the unique federation name. Therefore, the federation name either should be distributed by hand to all participants at start up or the federation name should be hard coded in federates. We picked out the second alternative.

The shutdown/termination of the federation is accomplished by the last federate, which resigns from the federation. The same rule applies here; all federates, while resigning, attempt to terminate the federation execution. The last one succeeds while others receive an exception because the federation still has members, and resign from the federation without terminating it.

2.5 NSTMSS COORDINATE SYSTEM

World Geodetic Coordinate System, the most commonly used coordinate system in navigation, is called the latitude (LAT), longitude (LONG), and height system. The Prime Meridian and the Equator are the reference planes used to define latitude and longitude. Latitude is measured through north and south, where as longitude is measured through east and west. This coordinate system is also used in most GIS tools.

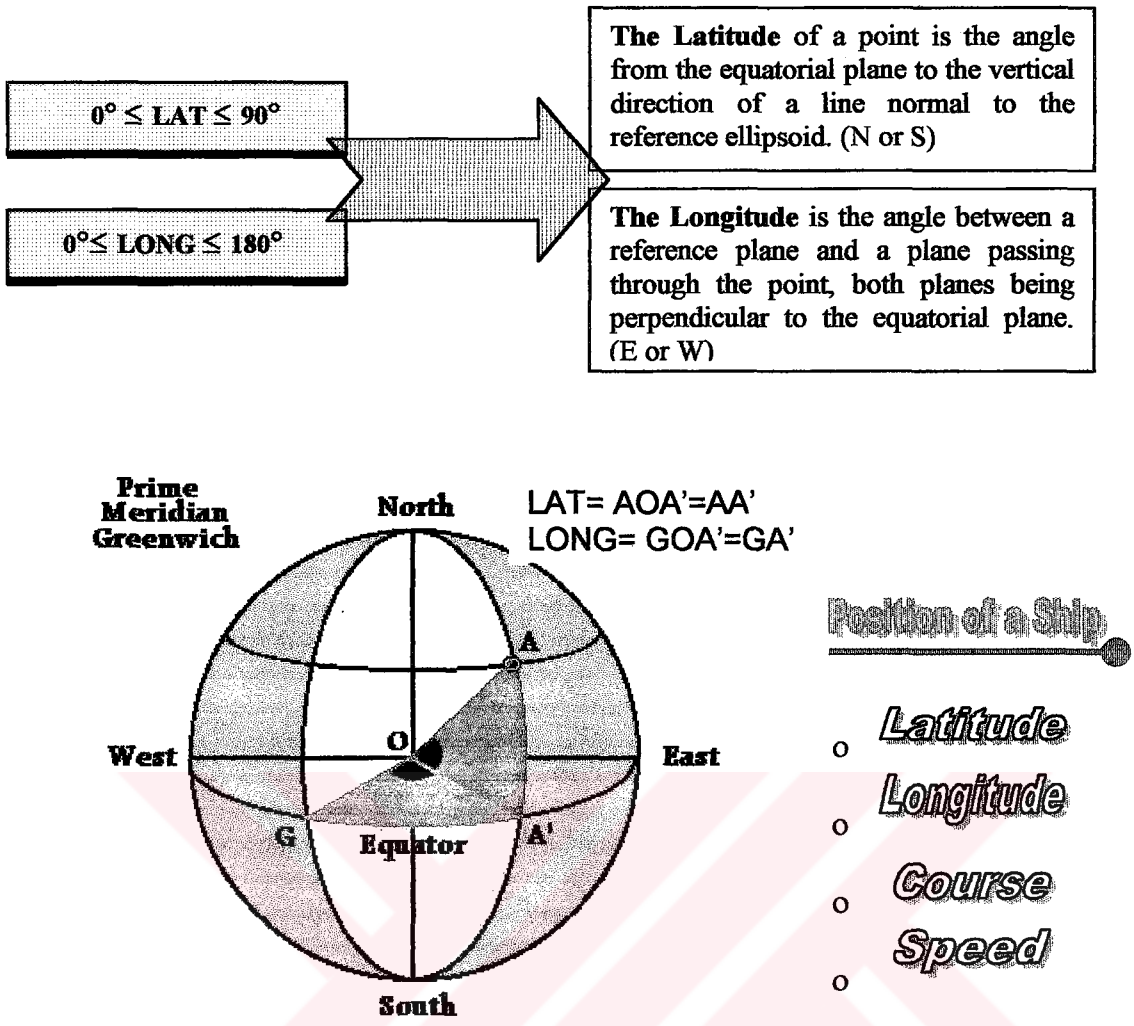


Figure2.10 Geodetic Coordinate System

In determining the position of a ship, mariners typically consider the latitude, longitude, course and speed to be a complete description.

Earth-Centered Earth-Fixed (ECEF) Cartesian Coordinate System, or called *Geocentric Coordinate System*, is also used to define three-dimensional positions. Earth-fixed, X, Y, and Z, Cartesian coordinates (XYZ) define three-dimensional positions with the origin located at an arbitrary point in the world. The system is called *right-handed* if it can be rotated so the three axes are in this position:

- The Y-axis, positive direction points toward the North Pole.
- The Z-axis, positive direction corresponds to increasing altitude.
- The X-axis completes a right-handed orthogonal system by a plane 90 degree east of the X-axis and its intersection with the equator. Positive direction corresponds to East.

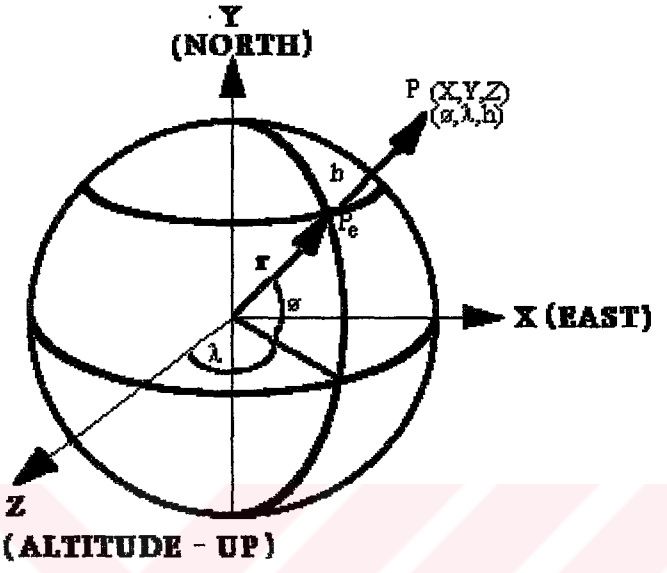


Figure 2.11 Geocentric Coordinate System

A ship position within geocentric coordinates is described in terms of its x, y, z coordinates, plus its orientation (course).

Environmental data are often naturally represented with respect to geodetic earth models. This implies an increased use of geodetic coordinates. Although, the geodetic coordinate system is the base coordinate system for NSTMSS, the geocentric coordinates are also used. The Mercator projection has straight meridians and parallels that intersect at right angles. Scale is true at the equator or at two standard parallels equidistant from the equator. The projection is often used for marine navigation because all straight lines on the map are lines of constant azimuth. The terrain data in GIS server or terrain database are geodetic coordinates, usually only LAT and LONG of a point is given. This provides the flexibility and capability to use charts with any scale used in navigation.

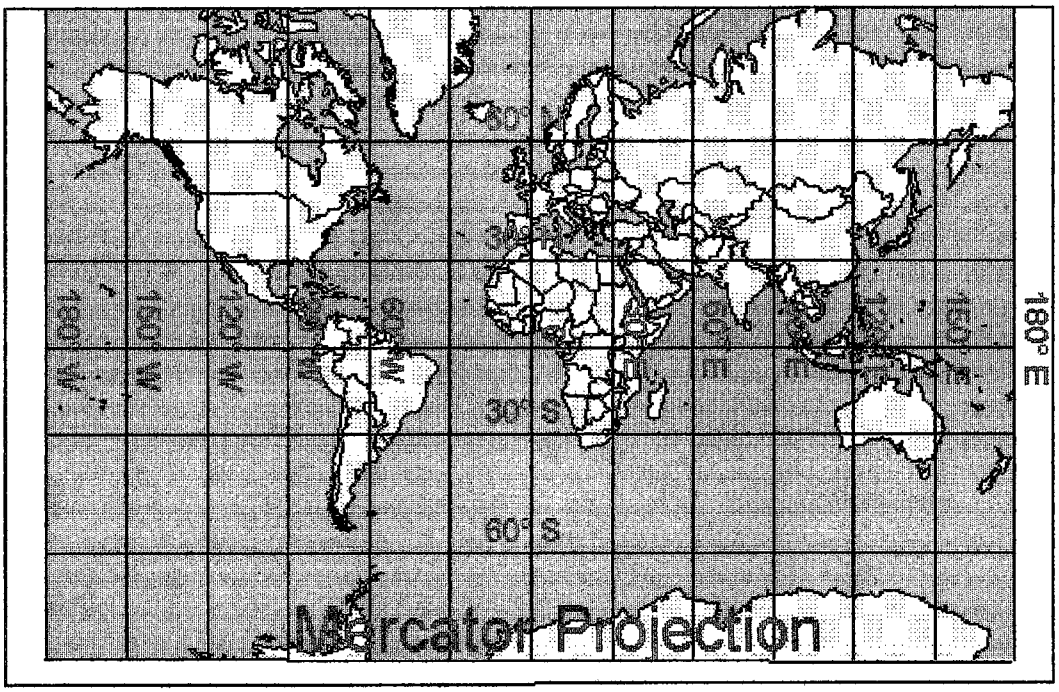
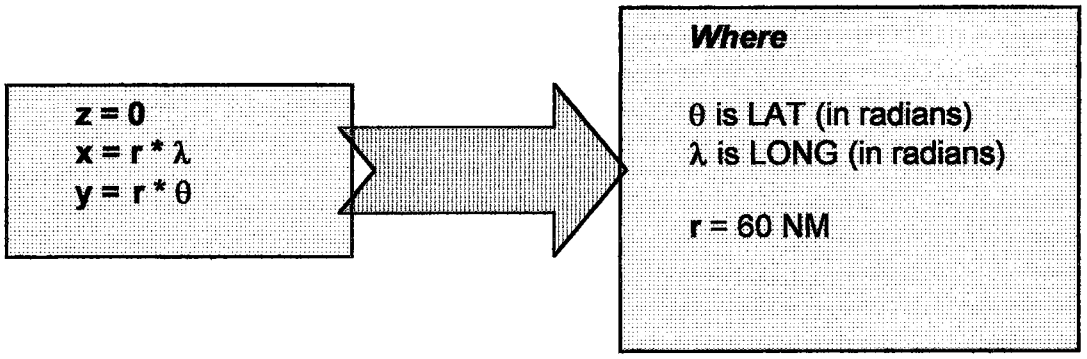


Figure 2.12 Mercator Projection

In NSTMSS, all federates know the positions of other federates in terms of geocentric coordinates (x, y, z, course, and speed). In KNOX and OTC federate, the global coordinate system must be converted into geocentric coordinate system because 3D graphical object position in the virtual environment is more simplified in geocentric coordinate system, since many graphical libraries use the right handed cartesian coordinate system.



The transformation between *geodetic and geocentric coordinates* can be thought as straightforward for illustrative purposes. Degrees of latitude can be translated directly into a linear displacement (1 degree is approximately

60 nautical miles), while the linear displacement associated with the degrees of longitude varies with latitude. This analogy is used for *illustrative purposes only*. The problem of transforming from geodetic coordinates to geocentric coordinates has received much attention for more accuracy [R98].

2.6 TIME MANAGEMENT

Within NSTMSS federation, Knox federate, OTC federate, and environment federate are both not time-constrained and not time-regulating according to the HLA time management terminology (a so-called "DIS-like" time management policy) [DMSO96]. Those federates advance their time independently. This also means that all objects and interactions are processed by RTI in the *receive order*. Receive order messages are placed in a queue when they arrive and are immediately eligible for delivery to the federate.

For example, Knox federate is a real time simulation and advances its time independently from other federates. It has a linear relation `dt=s*dw` where `dw` is a duration in *wall clock time*, `dt` corresponding duration in *simulation time*, and `s` is the scale factor, where `s=1`.

2.7 SUMMARY

The design of a federation has an important and long lasting impact on a distributed interactive simulation system. It is an agreement between federates and it prescribes the interaction rules. It defines the federation objects and interactions. The federation design has also effects over the reusability and interoperability. It reveals the capabilities of federates, time management schemes covered, and data distribution schemes.

This chapter gives a sense of the system at top level and RTI provided processes are also reviewed in this chapter. The details of components of the system; the federates are covered in chapter 3.

CHAPTER 3

NSTMSS FEDERATES

After reviewing the federation architecture, we can proceed into the details of the federates, which form the federation. This chapter covers the simulation entity federates: the Knox federate and OTC federate; the federation management federates: OSE federate; federation monitor federate; and environment federate. The following sections reveal the design and implementation details, explain the algorithms used, and give the details of the ship hydrodynamics. The following sections also explain the design and implementation of Tactical Message System.

3.1 KNOX FEDERATE (KNOXFd)

KNOX Federate (KNOXFd) implements a three-dimensional shiphandling simulator of a Knox-class frigate with a single user interface. KNOXFd provides the abilities for the Officer Of the Deck:

- To adjust the ship's course through the use of rudder by ordering a specific rudder angle,
- To adjust the ship's velocity by controlling the ship's engines.
- To learn the position of the ship in terms of latitude and longitude by looking at the indicators,
- To view the other ship echoes in the radar display,
- To look environment through the bridge,
- To receive/send messages from/to other ships or OTC.

In fact, these are the minimum requirements for the OOD to carry out a tactical maneuver in a surface task group. The abilities of OOD are thoroughly expressed in the form of use cases in appendix B. Many KNOX frigates can join into virtual environment. The ships are known by their names between users, though the federate internally know the ships with their object handle assigned by RTI executive. To name the ship at start up is the user's responsibility. If the user does not assign a ship name, the program automatically gives the default name.

3.1.1 KNOXFd MODULAR STRUCTURE

The internal structure of a KNOX federate is depicted in the figure 3.1. This is a modular structure to show the components of the federate at a top level.

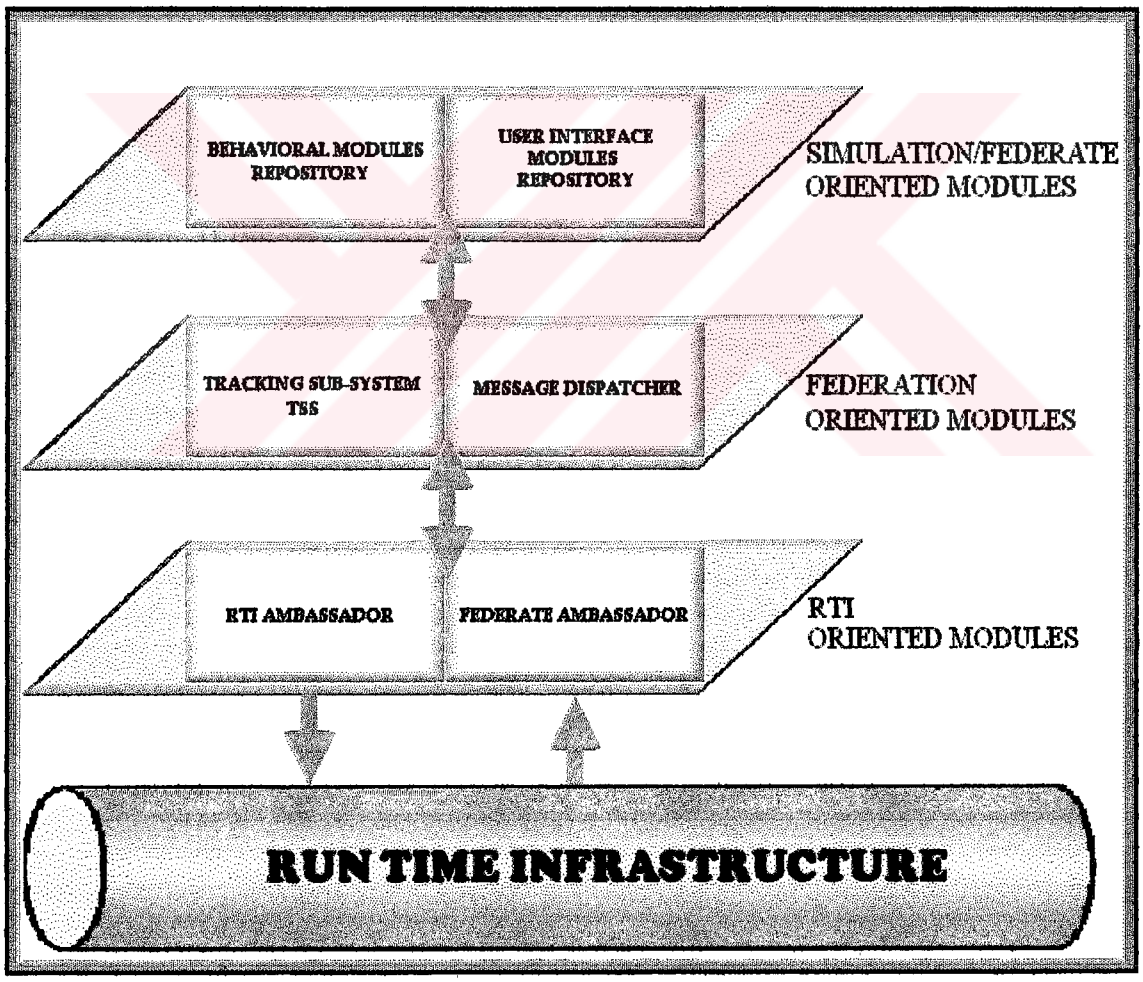


Figure 3.1 KNOXFd Modular Structure

T.C. YÜKSEKÖĞRETİM KURULU
DOKÜMANTASYON MERKEZİ

Simulation or federate oriented modules are related to the simulation capabilities of a federate. These modules have nothing to do with HLA or network concepts; they merely model and simulate some real life entities. In KNOXFd, these modules simulate the KNOX class frigate and her bridge equipment.

Federation Oriented Modules satisfy the federation-wide needs of the federate. These modules track other entities and filter the messages in the federation.

RTI-oriented modules are dedicated to communication related jobs. They are responsible communicating with RTI and receiving callbacks.
This layered architecture provides some advantages:

1. By separating the modules, reusability is enhanced both in design and implementation phases.
2. Encapsulation is achieved in architectural level. Each layer could be thought as a black box. The designer of one layer does not need to know the internal modular design of other layers.

For example, the simulation-oriented-modules layer simulate a KNOX class frigate warship; a designer can easily extract this layer and insert a new module group which simulates a submarine. He/she doesn't need to know the internal structure of either federation-oriented-modules or RTI-oriented-modules layer; it is enough to know the interfaces of the other two layers. As the second and third layers are re-used, the encapsulation and reusability are being achieved in architectural level.

SIMULATION/FEDERATE ORIENTED MODULES:

This layer encapsulates two sets of repositories: *'Behavioral Modules Repository'* and *' User Interface Module Repository'*. Figure 3.2 depicts the internal structure of the repositories.

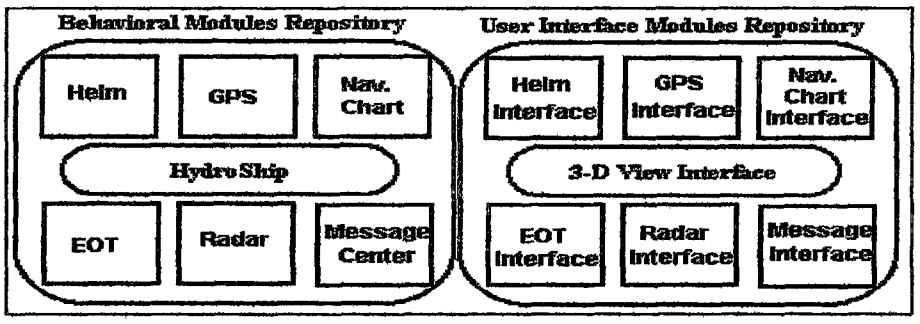


Figure 3.2 Simulation-Oriented Modules Internal Structure

Repositories are being developed to provide:

- Functions to allow the user of the ship, OOD, to control the ship. The OOD can see 3D views of back, ahead, right and left sectors of the ship, can control the helm and the Engine Order Telegraph (EOT), can use GPS (Global Positioning System) monitor and navigation radar.
- A reusable C++ class library for naval simulations, both behavioral and graphical.

The EOT and its interface module provide interfaces the user to control the ship's engines, while *the Helm module* and its user interface provide interfaces to turn the ship. The *GPS* module models and simulates a generic GPS monitor in the ship bridge. Message Center module enables the user to receive and send tactical messages to other authorities (e.g. to OTC, to other ships) in the virtual environment. *HydroShip* module implements simple hydrodynamics for the ship movement. It dynamically calculates the velocity and the heading of the ship in real time.

Although repositories are independent hierarchies, the module repositories of the federate are plug-ins to each other. The User Interface (UI) modules only interact with the user; they have no functionality. The functionality is achieved by the corresponding behavioral models. For example, `GPS_UI`, the user interface of the GPS, must know the position and velocity of the ship

to be able to display the values of these. While UI modules are being created, they are attached to corresponding behavioral models. Otherwise, they cannot be created. Therefore, a one-way communication channel, from user interface modules to behavioral modules is established. A *well-defined interface* is formed to achieve the communication. Thus, the UI modules know which behavioral model's interface they are. For example, while *GPS_UI* is being created, it knows that a GPS behavioral model exists and it is attached to that model, therefore it knows where it will get the values of velocity and position of the ship to display. A class interface is being depicted in Figure 3.3, using small circles connected with a line to the class in UML notation.

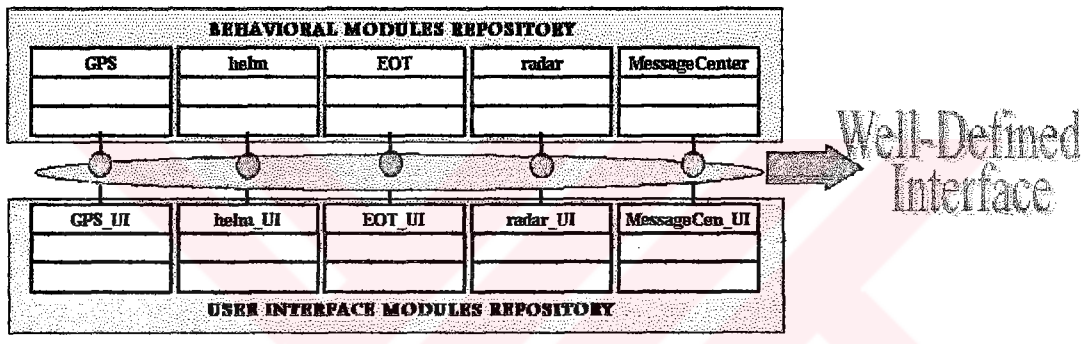


Figure 3.3 Repository Connections

By separating the repositories, we again increase reusability. For example, one can easily write a new GPS interface, which requires graphics programming knowledge. He does not need to know the internal structure of the ship, how the ship speed is calculated or the ship hydrodynamics. Thus he should focus on the user interface properties and graphics. User interface repository can serve many candidate GPS interfaces for the federate programmer to select from according to the desired detail of visualization.

Going from modular structure to the object oriented design, I map these two repositories into two class hierarchies. Figure 3.4 shows the class diagram for the federate oriented modules layer of the KNOX federate in UML

notation. The object oriented design of federates are discussed in detail in appendix-C.

The frigate class hierarchy has nothing to do with the user. It focuses on the functionality of virtual ship and its equipment. *The Single Graphical User Interface (sGUI) class hierarchy* interacts with the user and it is a bridge between the user and the simulation. The 'single' means that the ship interface is designed to be used on a single computer by a single user.

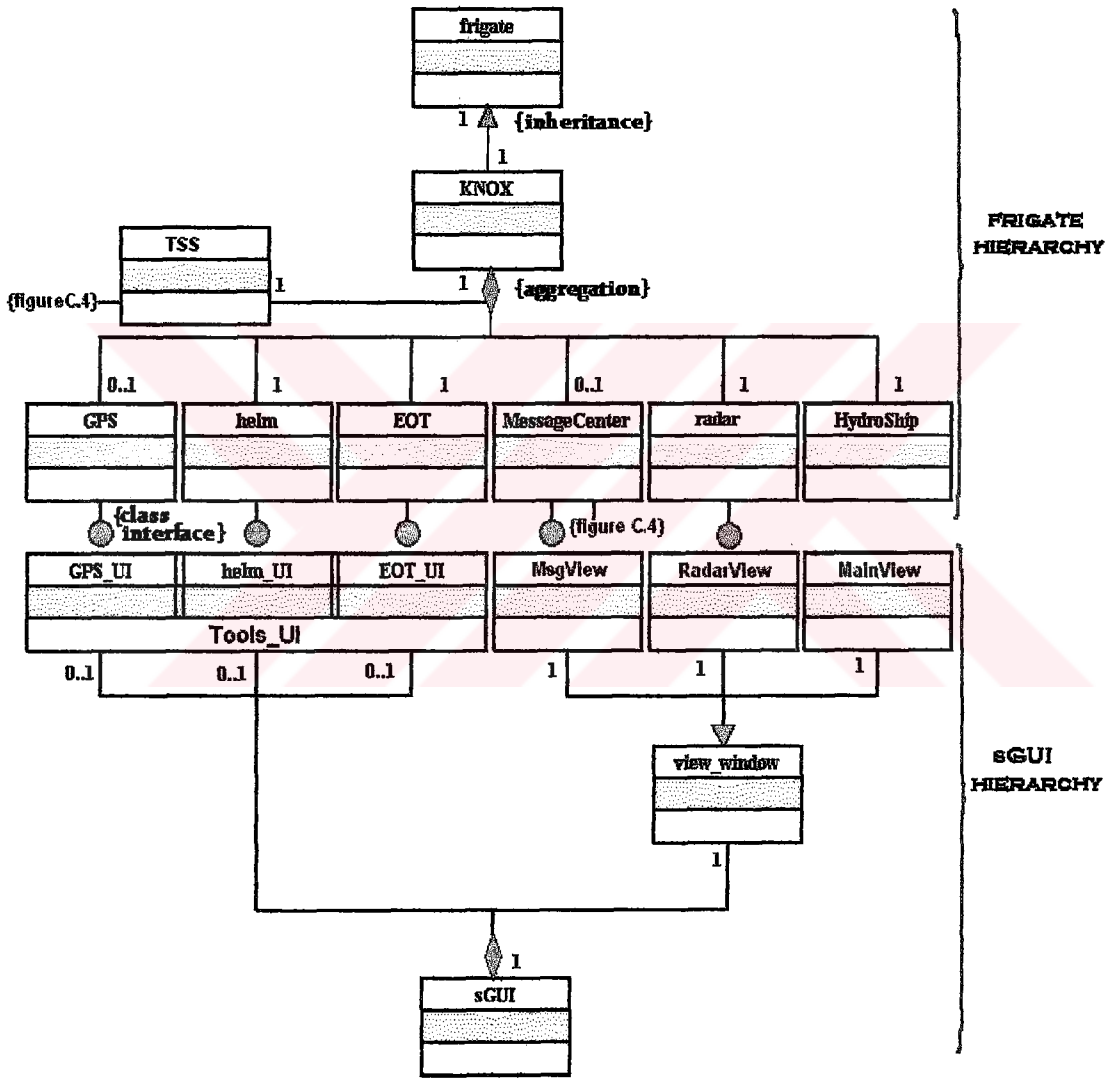


Figure 3.4 Frigate and sGUI Class Diagram

FEDERATION ORIENTED MODULES:

Federation oriented modules are composed of modules, which deal with the other participants in the common virtual environment. This layer consists of *Tracking Sub-System (TSS)* and *Message Dispatcher* modules.

Tracking Sub-System (TSS) is a module, which accurately represents the sense of common environment. This module implements a Dead Reckoning algorithm using the "players and ghosts" paradigm [GLKP94] to track other ships in the environment. It passes the dead reckoning data (position, speed, course) from other federates to the modules inside the repositories to render other ships at their positions. The dead reckoning algorithm, which is implemented, is expressed in following sections.

Message Dispatcher Module checks the receiver addresses for incoming messages. If the message received is being addressed to the local ship, than message dispatcher accepts the message and warns the message center. Message center and message dispatcher modules are being designed to give the ability to OOD to send and receive tactical messages. Message center module generates and displays the messages that will be sent or received.

RTI ORIENTED MODULES:

RTI-oriented modules provide services for communication between RTI and the federate. Federate-initiated services are invoked as an instance of *RTI Ambassador*. For example, when the ship changes her course, then the KNOXFd can inform other federates by using RTI Ambassador services. RTI-initiated services are invoked as an instance of *Federate Ambassador*. The Federate Ambassador "callback" functions provide a mechanism for the Federation to "signal" the federate. For Example, when an object is updated by another remote federate, this update is reflected by KNOXFd Ambassador to the local federate.

3.1.2 GRAPHICAL USER INTERFACE

In the field of virtual environments, one of the major areas of research has been the Human-Computer Interface (HCI). Although, the focus of this thesis does not include research in HCI, the reusable interface classes are being developed for KNOX and OTC federate. The user can easily interact with the system by using the keyboard and mouse. All the graphical interface functionality is being achieved by keyboard as well as mouse. The Figure 3.5 shows the KNOXFd main window interface.

CONTROLLING THE MOVEMENT OF THE SHIP:

In real life, watch team officers control the movement of their ships by uttering standard rudder and engine voice commands (explained at requirement analysis section) to personnel. But, in the Knox ship simulator, the OOD uses all the bridge equipment alone; this is why it is called single user interface.

The capability of interface, to control the ship's movement while viewing the scene, is achieved by placing controls in the control panel with the main viewing area, where they could be easily manipulated without distracting the Officer Of the Deck's view of the scene.

MAIN VIEWING AREA:

When designing the user interface of the ship simulator, the task of the modeling the movement of the ship was not the only problem. The ability to place the OOD onto a virtual bridge and allow freedom of movement among specific conning stations or viewing locations also needed to be considered.

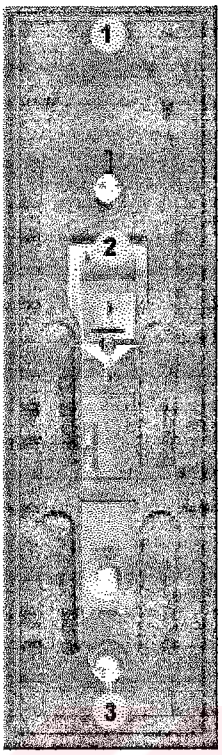


Figure 3.6 The OOD's positions

The approach to solving these problems was to attach the conning officer's viewing position to one of three possible locations on the ship model: the bow of the ship (1), the center of pilot house (2), or astern deck (3). The Figure 3.6 shows the three possibilities. Movement, in which the user is teleported between these positions, is accomplished through inputs from the CTI (command toolbar interface) or using the keyboard F1-F4 buttons. Upon program execution, the viewing location of the scene defaults to a location just forward of the ship's pilothouse. The OOD will see the foredeck of the ship as he/she would on a real ship along with the water and any landmarks that are directly in front of the ship.

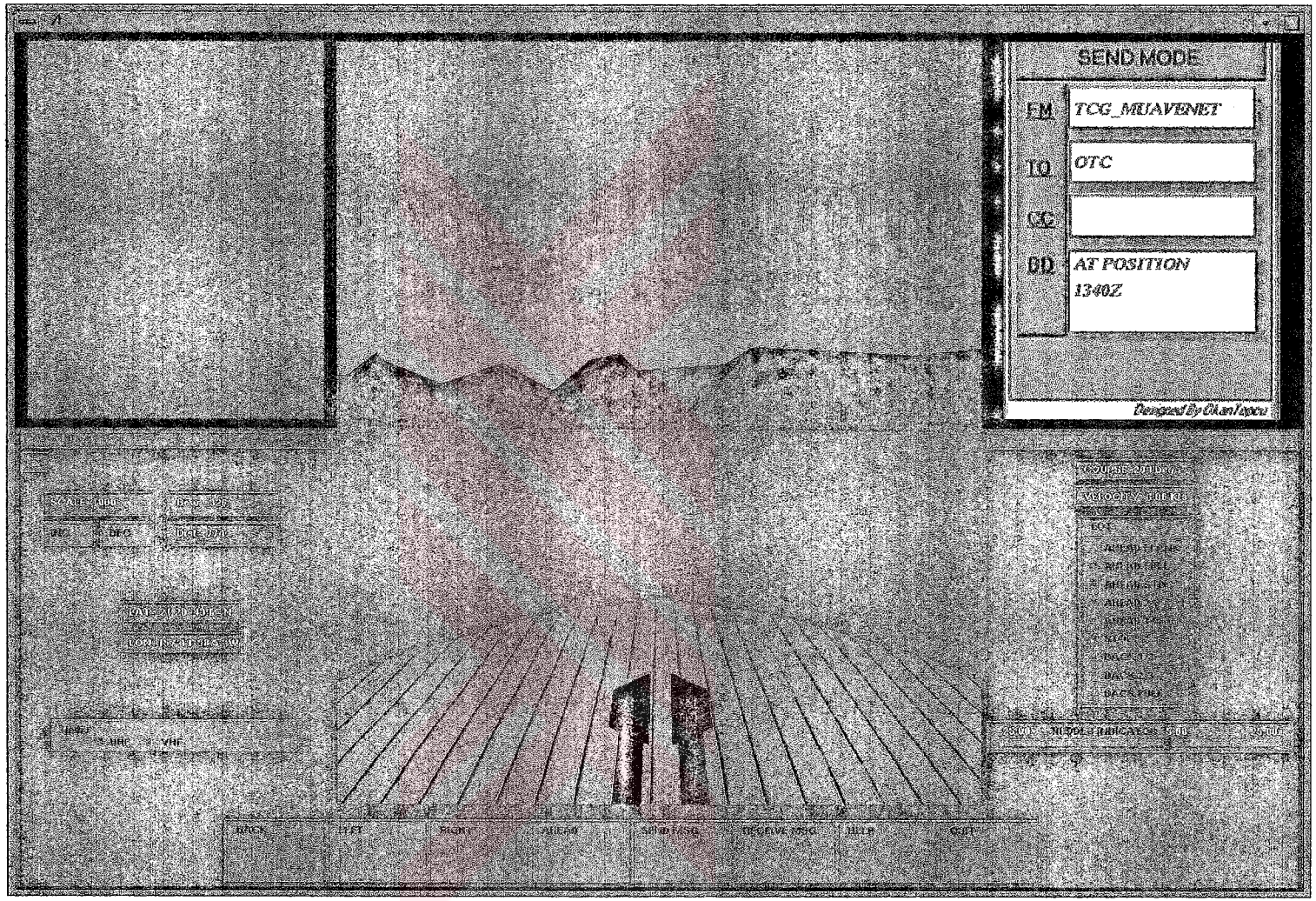
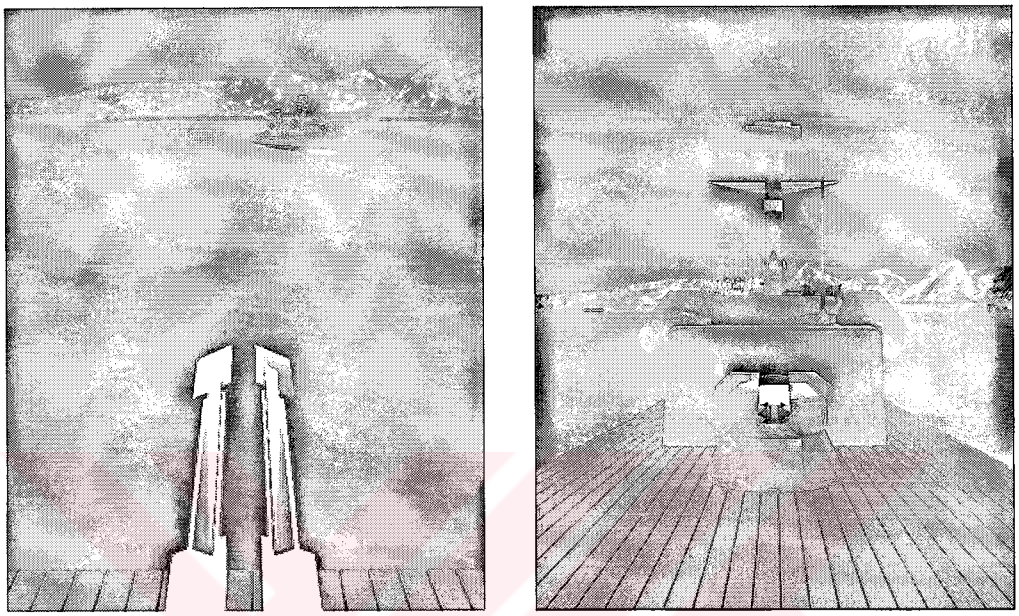


Figure 3.5 KNOXFd User Interface

In addition to moving on the ship, a form of head movement is being implemented where the Officer of the Deck could rotate his/her head to the desired viewing angle off the bow or raise and lower the view with respect to the horizon. Thus, by combining changes in viewing positions with head movement, the restriction of viewing objects that are close aboard is overcome (e.g. looking down into the water from the bow of the ship.).



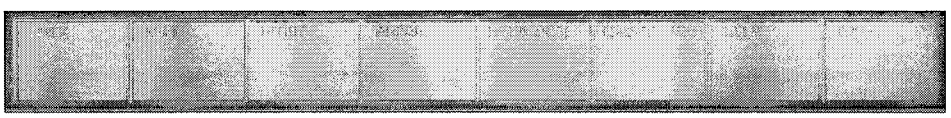
(a) (b)

Figure 3.7 Views from (a) the Astern Deck (location 3)(b) the Bow (location 1)

COMMAND TOOLBAR INTERFACE (CTI):

Command Toolbar Interface enables the user:

- To control the 3D views of the scene (right, left, ahead, and back sectors of the ship),
- To select between the send message tablet to write a new tactical message to other participants and receive message tablet to read an incoming message,
- To review the help file, which shows keyboard hot keys and their functions,
- To quit.



3.1.3 SIMULATION OF BRIDGE EQUIPMENT AND INDICATORS

There are numerous indicators and tools in the bridge some of which serve for the safety of navigation, and some serve for the military purposes. The simulated bridge equipment is chosen in a manner that the bridge officers often use these widgets in tactical maneuvering and formation steaming.

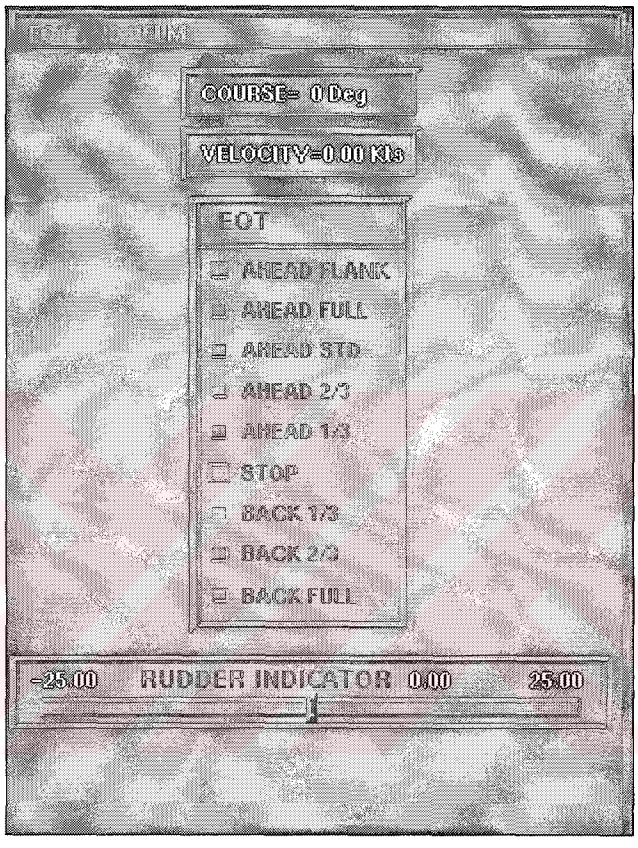


Figure 3.9 The Helm and EOT Panel

The OOD needs some vital information in order to position his/her ship correctly in the formation. This vital information minimally includes the ship's speed, course, position, and the distance and the bearing of other ships in the formation. All this information is being gathered for the OOD. The OOD also owns tools for controlling the ship's movement, for sending and receiving tactical messages, and for using the radar.

HELM AND ENGINE ORDER TELEGRAPH:

On just about every ship's bridge, the ship's speed, Rudder Indicator Degree (RID), engine order, and ship course are normally displayed on analog/digital devices located in various positions on the bridge, where the information they provide could be easily obtained without distracting the conning officer from his/her desired view of the surrounding scene.

The helm and engine order telegraph can be easily used in the panel to control the ship's movement. When the helm is being controlled by using the keyboard keys, rudder indicator degree steps are five degrees, in like real rudders. But, when it is controlled by mouse, rudder indicator degree steps are 1 degree. Thus you can "utter" rudder commands such as "*right 13 degrees rudder*", which you cannot in real life.

RADAR:

Aboard ship, radar plays a major role in gathering information and in maneuvering and navigation. Although the developed radarscope does not look like a real radar repeater view, it has some essential functionality, which all navigation radars provide.

It provides a God's view of the geographic area, which conventional radars provide as green beams. The radar center is attached to bridge, pointing always to North. Thus when the ship moves, the covering area also moves. The OOD can change scale (covering area) between min. and max. ranges by zooming in/out.

Second, it shows the objects around as iconized, unlike a real radar which shows them as echoes. Our radar detects all surface ships in its coverage area. The radar display the distance and true bearing of an echo, which is very important in formation steaming. By the help of this functionality, we can check out and keep correct ship's position in formation.

GPS:

GPS is one of the principal indicators in the bridge. Its main functionality is the ability to display the ship's position in terms of latitude and longitude. It shows the values in a readable open format: "degrees minutes seconds" (e.g. 123° 45′ 13″ W). GPS indicator is placed into radar panel.

MESSAGE CENTER:

Tactical communication is very important in tactical maneuvering. The OTC orders which formation will be carried out, what the formation speed will be, what the ship sequence is etc. by *sending messages* to ships. A message handling system is embedded in user interface. Thus, the OOD is able to read incoming messages and write new messages to send. The Figure 3.10 shows the message interface. The user switches between the send and receive modes by using CTI or keyboard F7 and F8 keys.

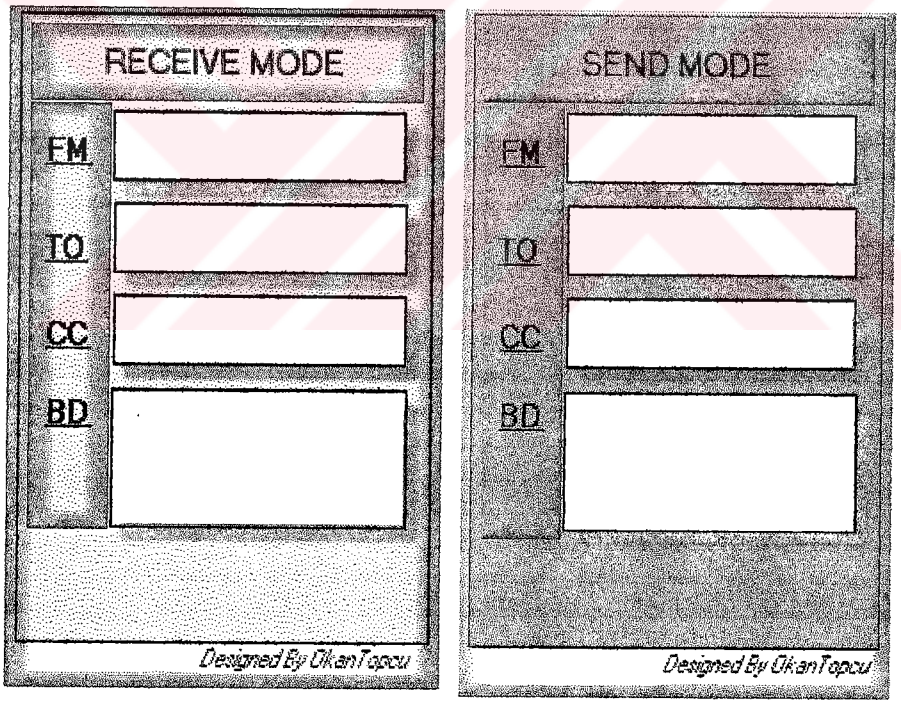


Figure 3.10 Message Interface

The user can only write new messages while in send mode. When a new message is being received, the message center informs the user and the

user should switch to the receive mode in order to read. The implementation of Tactical Message System is explained in the OTCFd section.

3.1.4 HYDRODYNAMICS AND MANEUVERING

The field of modern marine vehicle dynamics goes back to the year 1737,where it has its roots in Euler's theory of motions in still water [B62]. In this theory, Euler likened the oscillatory motion of a floating vessel to a pendulum, ignoring any effects of water resistance or other external forces acting on the vessel.

FORCES ACTING ON A SHIP:

The movement of a ship is caused by forces acting upon its body coordinate system. These forces consist of external forces such as ocean current, prevailing winds and wave action (or sea-state). Internal forces are those forces generated by the ship's propulsion machinery (e.g. propellers) and control surfaces (rudders). Internal and external forces can be further classified into three categories – *controllable forces, semi-controllable forces* and *uncontrollable forces*.

Controllable Forces are described as those forces that can be controlled by the actions of the bridge officer through the use of the ship's machinery (e.g. rudders and propellers). The ship's propellers generate *thrust* that moves the ship in either a forward or backward direction, while the rudder generates a component of the thrust, thereby creating a moment or *torque* about the center of gravity. The effect of this torque causes the aft end of the ship to swing either to the port (left) or starboard (right) side. It is these forces generated by the propeller/shaft combinations that must be understood to steer the ship and present a good training mechanism for the ship simulator.

Semi-Controllable Forces are those forces that are generated by the surrounding area that the ship is located in. These forces are usually present in narrow waterways where thrust generated by the ship's propellers and the wake resulting from the ship's movement through the water cause the

immediate surroundings of the waterway (e.g. sides, bottom...etc) to generate repelling forces which act against the ship.

Uncontrollable Forces are those caused by nature such as winds, currents and waves. Conning officers have no control over these forces but must either compensate for them or use them to their advantage.

In the scope of this thesis, we model the controllable forces only; that is the effect of propellers and rudders, although the other two forces acting upon the ship, are important.

SIMPLE HYDRODYNAMICS MODEL:

Prior to developing a hydrodynamics model for the shiphandling simulator, the characteristics of the real environment that the simulator represents were examined to determine which behavioral characteristics of the ship could be eliminated with the absence of certain uncontrollable and semi-controllable forces such as high winds and large waves in the operating environment. The results of this analysis determined that the ship would be operating in calm seas, which there are no wave, current, and wind. Therefore, we can consider heave and sway as negligible, as well roll and pitch rates.

By making this assumption, we conclude that we focus on controllable forces to provide shiphandling characteristics, such as *ship's turning diameter*, which is deeply affected by ship's heading and speed that are the result of thrust, and torque, since this is a simple simulator. The other objective is to achieve an adequate rendering performance by avoiding complex arithmetic computations.

The simulated ship has a mass (derived from weight), she has one propeller that is located laterally through the centerline and it has a rudder located longitudinally a fixed distance from the centerline. Using these characteristics and assumptions, I chose a simple hydrodynamics model of controllable forces developed in NPSNET simulators [ZPFB93, NG95]. For the purpose

of not dispersing the focus of thesis, I will not present the formulas of theory of ship hydrodynamics (for the theory refer to [MS65, B62]).

LINEAR MOTION:

The ship's propellers generate thrust that moves the ship in either forward or backward direction. Thrust from propellers defines the velocity of the ship.

```
F=m*a
a=F/m
F=Total thrust
Thrust=constant (newtons/RPM)*RPMs
V=dt(F/m)
```

Thrust generated from propeller is obtained by multiplying the shaft speed by a ship dependent constant. By the derivation of acceleration, we obtain velocity of the ship.

ANGULAR MOTION:

When rudder is deflected by a certain amount of angle, the trust from the propeller causes the rudder to generate a rudder torque. The *Rudder Torque* is figured out by multiplying the trust and sine of *rudder deflection*. When the rudder deflection is zero the aft of the ship tends to swing right or left because of the turning of the shaft. *Shaft Torque* is defined by multiplying the trust and lateral distance, which is the distance between shaft and center of gravity. *Total Torque* is the sum of rudder and shaft torques. From total torque, we find *angular velocity* and finally we find *new heading* by the sum of old heading and angular velocity with respect to the time [MS65].

3.1.5 BASIC PROGRAM FLOW

In chapter 2, the basic program flow of federates are divided into three phases: system initialization, main simulation loop, and system shutdown.

SYSTEM INITIALIZATION:

System initialization phase contains three sub-phases: user class initialization, Performer initialization, and RTI initialization. Figure 3.11 summarizes the steps.

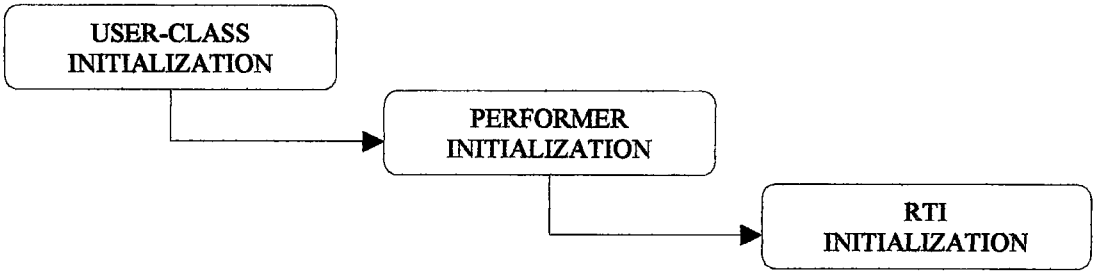


Figure 3.11 System Initialization

At *User class initialization* step, the objects of `sGUI` and `frigate` classes are being created and their references to each other are set. The command line interface is being interpreted. The main attributes of the ship object (e.g. name, course, velocity, and position) are set to its default values. The flow of every Performer program begins with a set of Performer specific initialization functions. At the *Performer initialization* step, the Performer specific initializations include (They are bulleted according to program flow):

- The allocation of shared memory (this shared memory is for federate specific works, nothing to do with network or other federates),
- Setting and configuring multiprocessing mode (Performer allows the use of multiprocessors),
- Setting of graphics pipeline.

Besides those compulsory initializations;

- Main scene graph is created and constructed (explained in following section 3.2.6),
- Message center scene graph is created and constructed (explained in following section 3.3.3),
- Window is created and initialized (e.g. window name is set),
- Channels are created and initialized (e.g. the size and position of channels are set),

- The bridge equipment displays are initialized (e.g. the velocity display is initialized).

RTI initialization step includes creation of federation, joining to the federation (expressed in chapter 2), creating regions, publishing and subscribing of federate capabilities, and registering objects and interactions that the federate is capable of.

MAIN SIMULATION LOOP:

Main simulation loop includes a loop that continuously calls the pre-frame, frame, and post frame stages. The figure 3.12 shows the steps of main simulation loop.

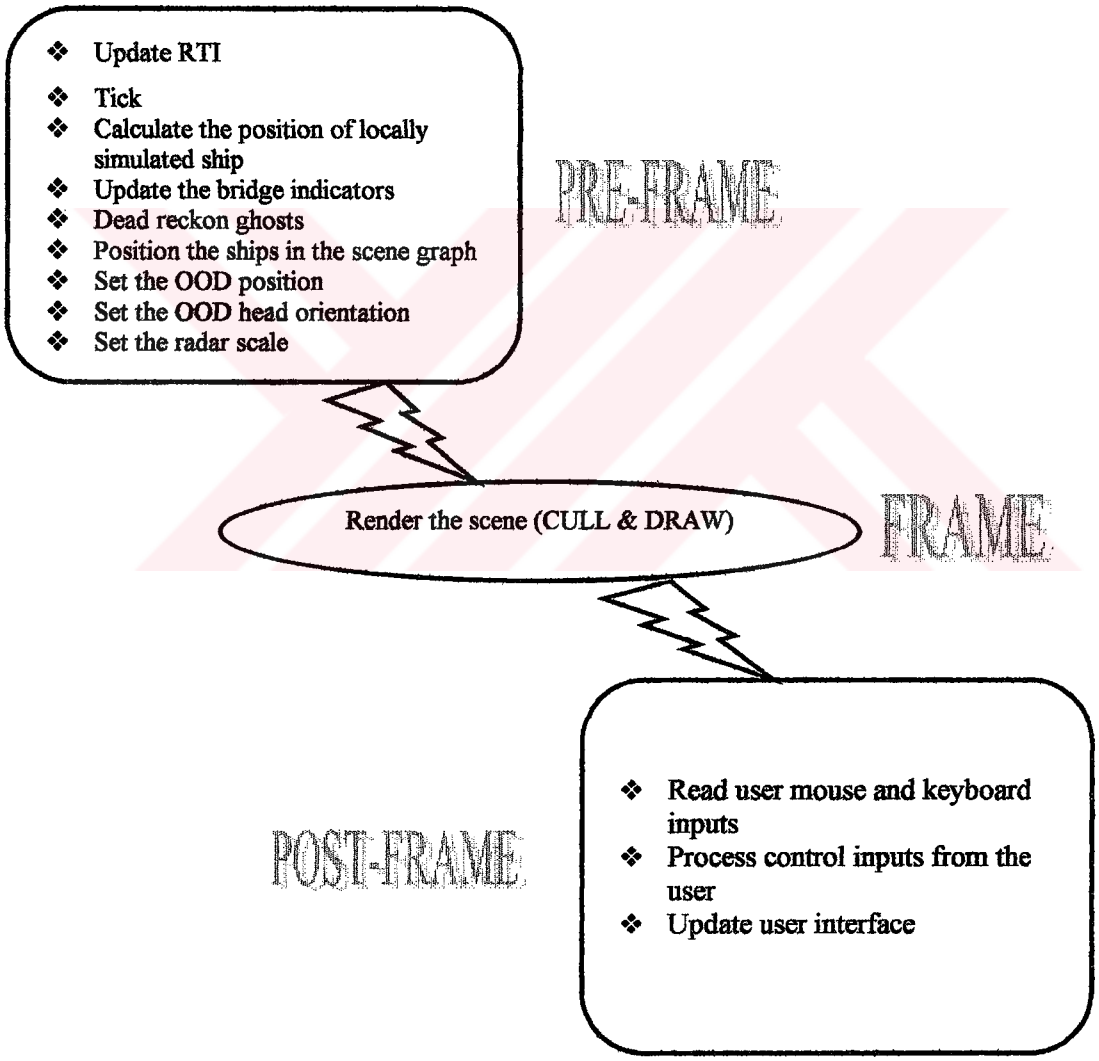


Figure 3.12 Main Simulation Loop

Let's review and explain the pre-frame stage:

- Update RTI (broadcast its own state to network),
- Tick means reading the callbacks and the incoming events,
- Calculate local ship's velocity, course and position based on the Engine and helm status,
- Update bridge equipment displays. The values of the displays are those that are calculated at the preceding step,
- Dead reckon ghosts. The Tracking Sub-System (TSS) holds a list of the ships in the common environment and calculates their positions according to the dead reckoning algorithm.
- Update the transformation and rotation nodes in the scene graph. The geometry repositioned within the scene.
- Calculate the OOD position. In fact, we set the position of the camera. The position of the camera (the OOD's eyes) is not fixed. The first calculation is for the global position of the camera because the camera moves as the ship moves. The second calculation is for the position of the camera relative to the ship.
- Set the OOD head orientation. Remember that the OOD can rotate, raise, and lower his/her head.
- Set the radar scale, if radar scale has changed.

After all the necessary calculations done in pre-frame, we can render the frame. The frame phase consists of two stages: *CULL* determines which geometries in the scene are visible (in the viewing frustum). *DRAW* renders all visible geometries. Each stage runs as a separate traversal over the scene graph.

At the post-frame stage, the snapshot of user input queues, both from keyboard and mouse, are captured and handled. The GUI is being updated based upon the user inputs.

SYSTEM SHUTDOWN:

System shutdown phase consists of three stages: Performer shutdown, user class shutdown, and RTI shutdown. Performer shutdown is necessary to terminate parallel processes and to release the shared memory. At user class shutdown stage, the created RTI objects and interactions are deleted and other federates are informed. At RTI shut down stage, the federate resigns and tries to destroy the federation.

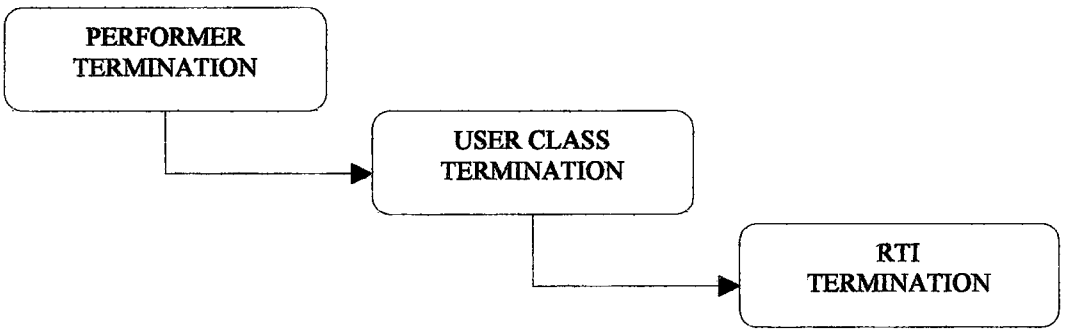


Figure 3.13 System Termination

3.1.6 SCENE CONSTRUCTION

The visual runtime database is a directed, acyclic graph rooted at a node called scene node. A scene is rendered into a channel or several channels, which are culled and drawn in the graphics pipeline, as shown in the Figure 3.14. The pipe is the software abstraction of the hardware graphics pipeline.

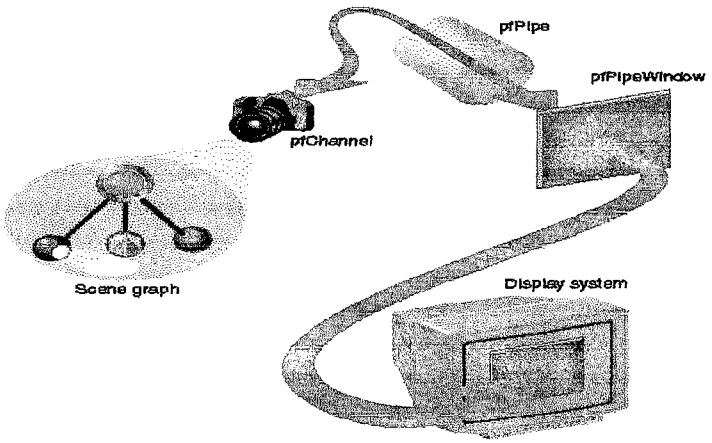


Figure 3.14 How Performer Renders a Scene Graph [SGI97]

The hierarchical visual database is being constructed logically grouping the entities in the scene and depicted in the figure 3.15. The figure only depicts the main window scene graph hierarchy; the message center scene hierarchy is expressed later. `mwScene` is the root of the scene hierarchy, which is rendered. Hierarchy is divided into two logical groups at start. Objects that are linked to root with thick lines are *statically added* to the `mwScene`. Objects that will be linked to root, and shown with dotted lines are *dynamically added group*.

"*Statically added*" means that the nodes in this group are added at program start up. For example, when Knox federate is started, the main scene graph is only constructed with the nodes in this group (the sun and the ship geometry). The lifecycle of those nodes are the same with the program lifecycle. On the other hand, the virtual environment is dynamic in the sense that there are many federates joining and leaving at run time. When a new ship joins the distributed environment, KNOX federate ambassador callback discovers the new created federation object and triggers the functions, which are responsible for adding and removing the `surfaceShip` group into scene. Therefore you should modify the scene graph at run time. So the group of nodes that are added or deleted at runtime are called *dynamically added group*. When a ship is changed its course for example, the updated ship is being found in the scene graph and its hpr (heading, pitch, and roll) values is set by the help of the `shipDCS` node, which is a transformation and orientation node.

Dynamically added group is also divided into two logical groups: *Stationary Group* and *Moving Group*. The stationary group contains stationary objects such as the geometry of the geographic region (e.g. island) or the geometry of the stationary objects in this geographic region (e.g. lighthouses). Environment federate informs other federates about the geographic area in which the scenario takes place. The moving group consists of moving objects in the environment such as other ships.

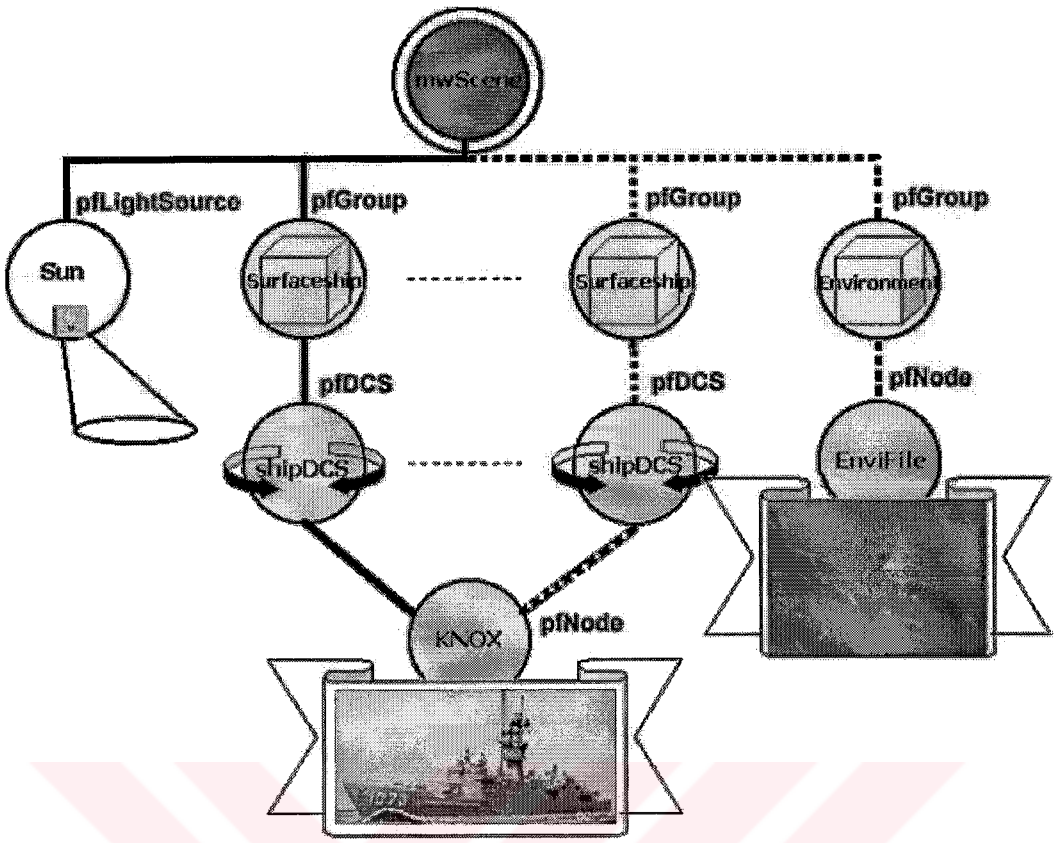


Figure 3.15 Main Scene Graph

3.1.7 ENVIRONMENTAL EFFECTS

We make the following assumptions for the present version of NSTMSS:

- ❖ Sea is always in the quiescent state.
- ❖ Air is always in the quiescent state.

3.2 OTC FEDERATE (OTCFd)

3.2.1 OTCFd MODULAR STRUCTURE

OTC Federate (OTCFd) is designed to model formations and maneuvers at the Tactical Level. It provides bird's eye view of all the warships and the operational area. OTCFd provides interfaces to user OTC to control and

order the formations of ships to achieve a given duty. Internal Structure of OTC Federate is depicted at the Figure 3.16.

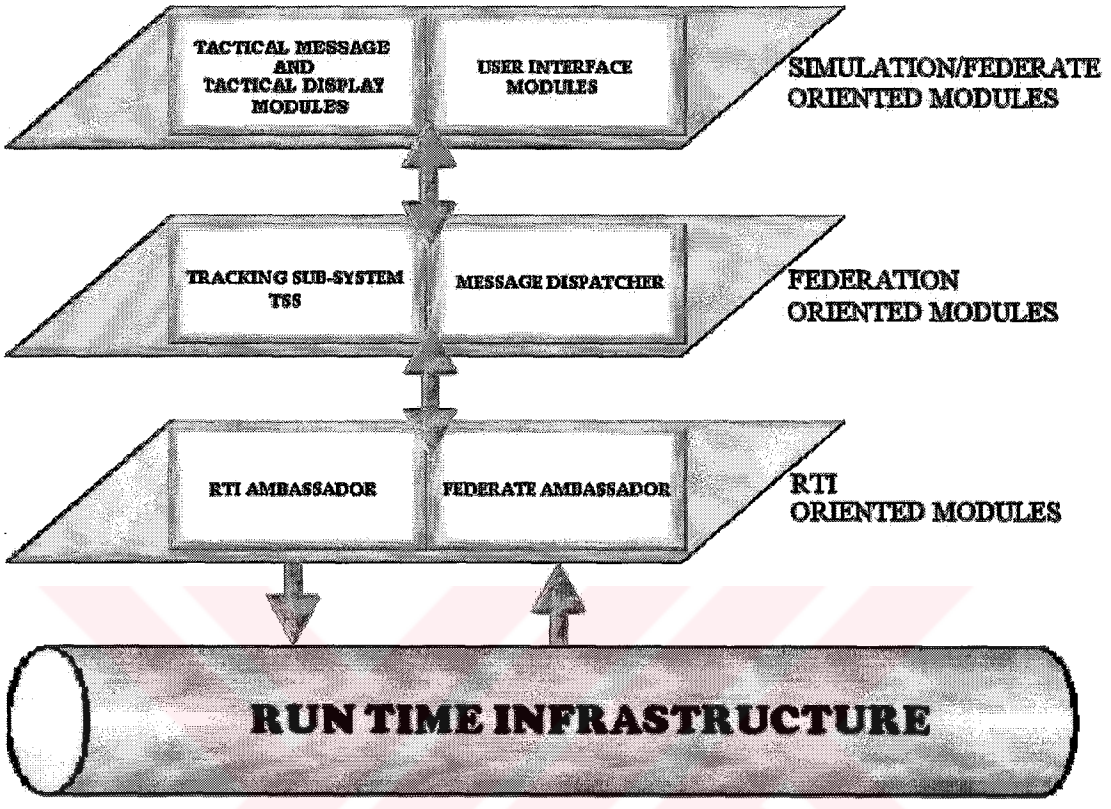


Figure 3.16 OTCFd Modular Structure

The modular design of OTCFd is similar to the KNOXFd modular structure. Time takes for the design and development of OTC federate is significantly reduced by the help of reuse of KNOX federate at design and implementation level.

- The design of federation oriented modules layer and RTI oriented modules layer is reused
- The implementations of the modules in those layers are also reused with some modifications on the code.

The functionalities of federation oriented modules layer and RTI oriented modules layer are the same with KNOX federate. Tactical message and tactical display modules are expressed in following sections. The class hierarchies are given and explained in appendix C.

3.2.2 GRAPHICAL USER INTERFACE

The Figure 3.17 shows the OTC interface. Graphical user interface is designed to meet the requirements:

- To scroll up and down the tactical display,
- To scroll right and left the tactical display,
- To zoom in and out,
- To send and receive messages,
- To read the geodetic coordinates.

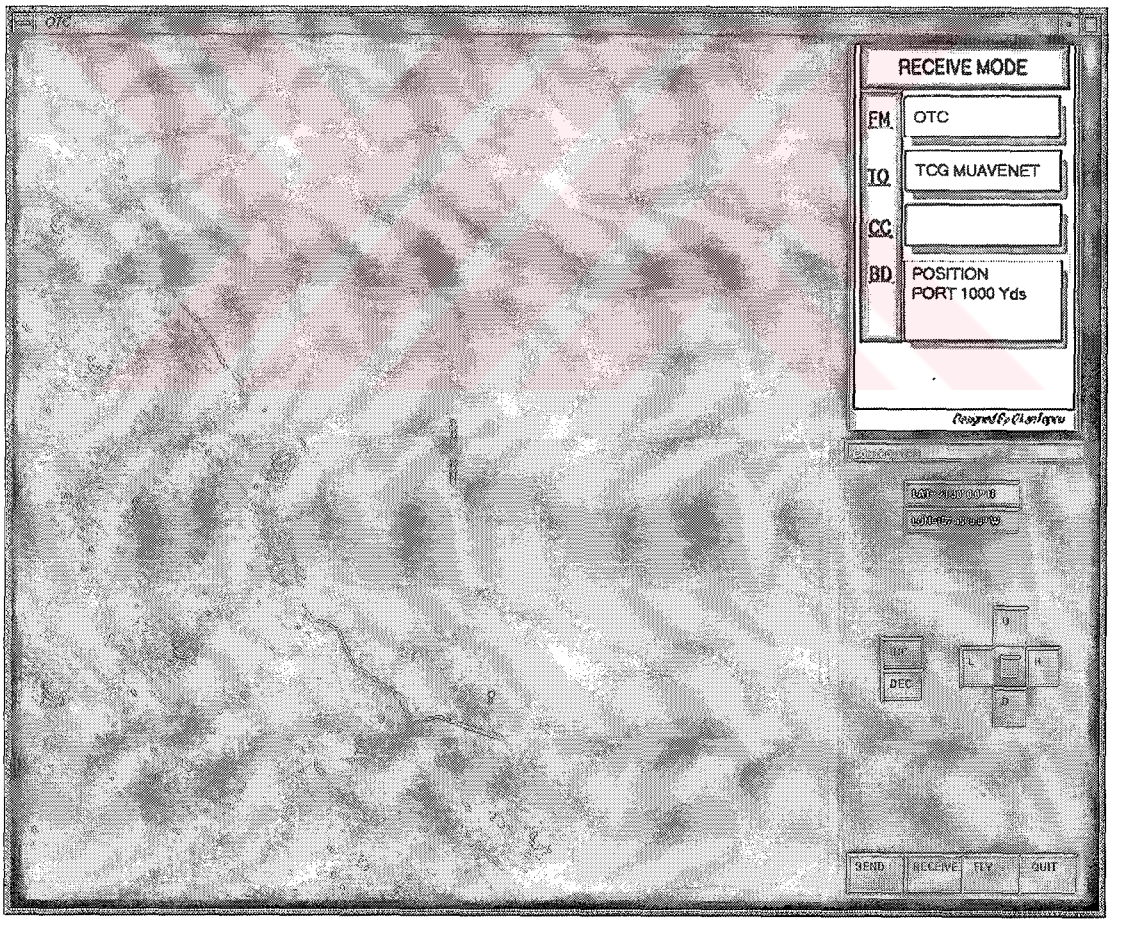


Figure 3.17 OTC Interface

3.2.3 TACTICAL MESSAGE SYSTEM (TMS) MODELING FOR FEDERATION

Tactical maneuvering orders are conducted via an established communication network, which is called *Tactical Communication Network.* The whole communication occurring in this network is called Tactical Communication. Although this is a voice communication, it has been modeled as a written-message interface in NSTMSS and called *Tactical Message System (TMS).*

Tactical Message System deals with all the communication messages between the commodores (OTC) and captains (OODs). TMS describes the message formats, defines the senders and receivers. TMS handles four message types: weather reports (WeatherRep), task orders (TaskOrd), formation messages (FormMsg), and status reports (StatRep). Explanations and examples of the message formats are given in appendix A.

CLASS STRUCTURE:

The design of the class structure is depicted in the figure 3.18(a) and the run time linkage between objects is depicted in collaboration diagram in Figure 3.18(b).

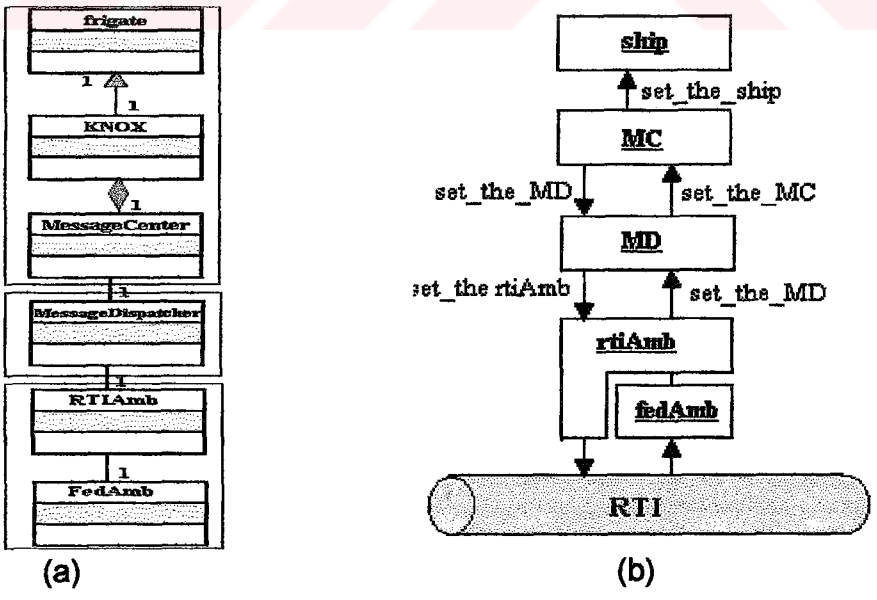


Figure 3.18 (a) Class Diagram (b) Collaboration Diagram

The links are used to form communication channels to trigger the objects. The triggering system is showed in the "sending message sequence diagram" illustrated in the Figure 3.19. The OOD triggers the system by pressing "enter" key to send the message. The detailed explanations are being made in appendix B-Object Oriented Analysis.

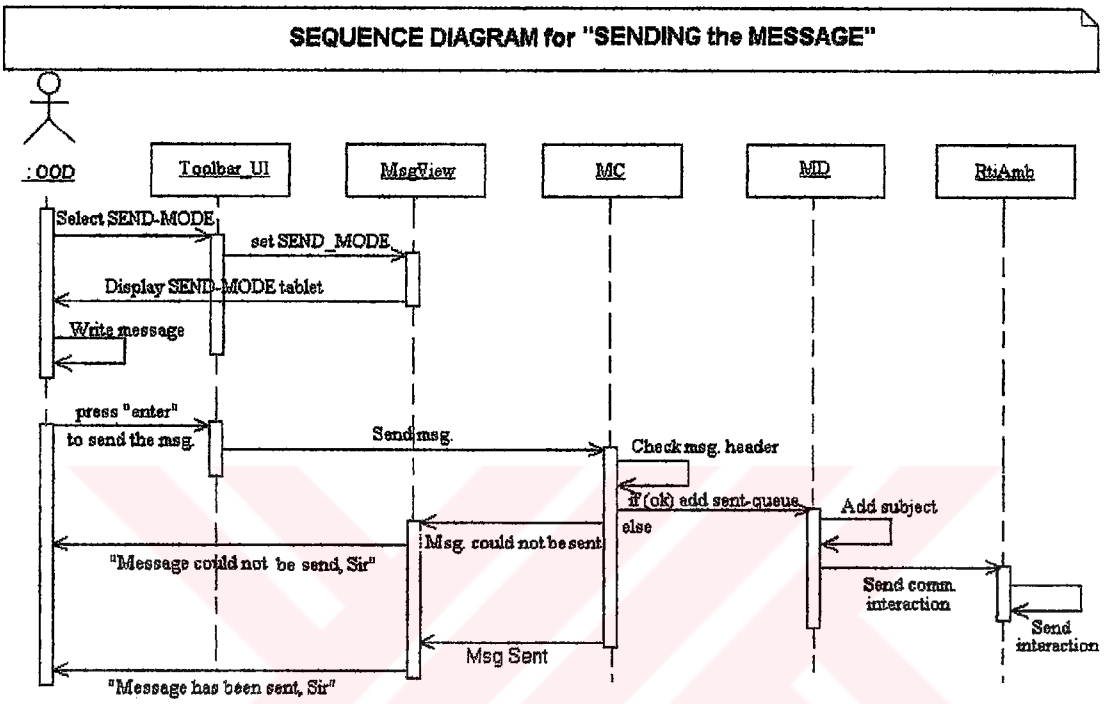


Figure 3.19 Sending Message Sequence Diagram

Message Center is responsible to check the message format prepared by the OOD. For example, it checks whether the sender name filled in is the same with ship name. *Message Dispatcher* is responsible to check the headers of incoming messages. If the incoming message header contains our ship name, the message is accepted and the user is informed that a new message is received. *RTI Ambassador (RTIAmb)* and *Federate Ambassador (FedAmb)* classes handle the network communication.

DATA DISTRIBUTION MANAGEMENT (DDM):

The RTI effectively serves as an intelligent switch – matching up producers and consumers of data, based on declared interests without knowing details

about the data format or content being transported. Data Distribution Management (DDM) facilities of RTI provides an effective mechanism to extend the RTI's switching capabilities. RTI allows us to define *routing spaces* within federation. The routing space is a collection of *dimensions*. A dimension is an attribute. The dimensions are used to define *regions* [DMSO98b].

The Figure 3.20 illustrates such a routing space. Formation messages are broadcast in *Ultra High Frequency (UHF)* bandwidth, where tasking orders are broadcast in *High Frequency (HF)* bandwidth. One dimensional communication space is created to handle the tactical communications. *Communication Space* is the name of routing space, where has one dimension called *Frequency Spectrum*. Two regions are created: *UHF region* for the communication of Formation Messages and *HF region* for the communication of Tasking Orders. For DDM to support arbitrary dimensions, a generic representation scheme is needed to express extents. The full range along a given dimension is mapped onto the interval *[MIN_EXTENT,MAX_EXTENT]*. For example; a value on the dimension *Ultra High Frequency Region* must be mapped to a number in the range of *[MIN_EXTENT=0,MAX_EXTENT=100]*.

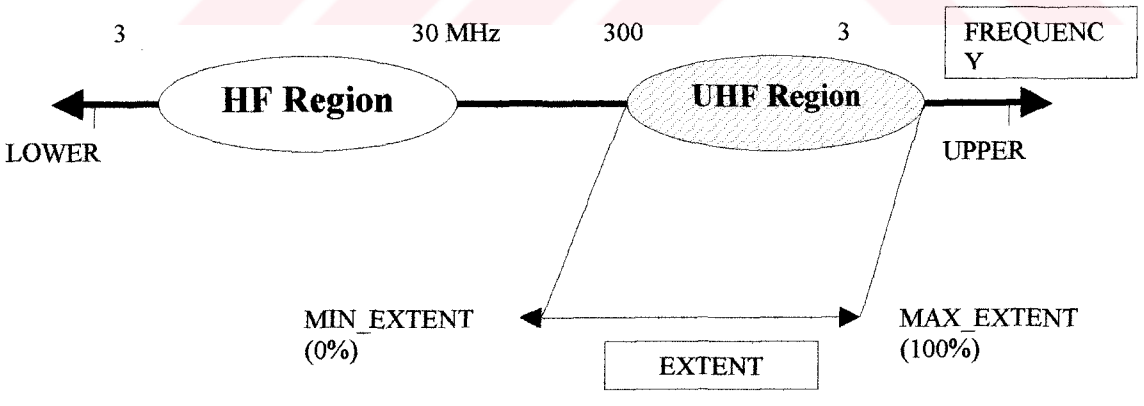


Figure 3.20 NSTMSS Communication Regions

The routing space must be declared in the federation execution details (FED) file. NSTMSS FED file includes parameters that identify the routing space and the dimensions.

```
(spaces
  (space CommunicationSpace
    (dimension FrequencySpectrum)
  )
```

Federates are responsible to create regions on a specified routing space. The action synopsis of a federate creating UHF region is that the federate owns a UHF radio and has the ability to send and receive messages in UHF bandwidth. The following pseudo-code snippet explains how a federate creates a region:

```
Create Region
{
      Get Routing Space Handle from RTI;
      Get Dimension Handle from RTI;
      Create Region using RTI services;
      Set range lower and upper bounds of region;
      Let the RTI know the created region;
};
```

The communication interests of a federate should include regions. The subscription and publishing of FormMsg interaction must be bound to UHF Region. This will turn on the UHF radio. Either a federate subscribes to FormMsg interaction without a UHF region; it will not be able to receive the messages because it has UHF radio but in off state.

The federate is now on receive-mode in UHF Region and can receive any message sent through UHF bandwidth. While sending the message, the federate should explicitly state the region with the call: *sendInteractionWithRegion()*. The complex program codes dealing with data distribution management is encapsulated in RTI and federate ambassador classes.

Events and requests that are not associated with a particular routing space are automatically communicated in the default routing space. Remember that the other NSTMSS objects are also communicated in default routing space.

INTERFACE IMPLEMENTATION DETAILS:

Message Center interface provides a user interface to read and to write messages and it is shown in previous sections in the Figure 3.10. The interface is being implemented as two rectangular tablets attached a texture on it. A scene graph depicted in the figure 3.21 has been designed and implemented.

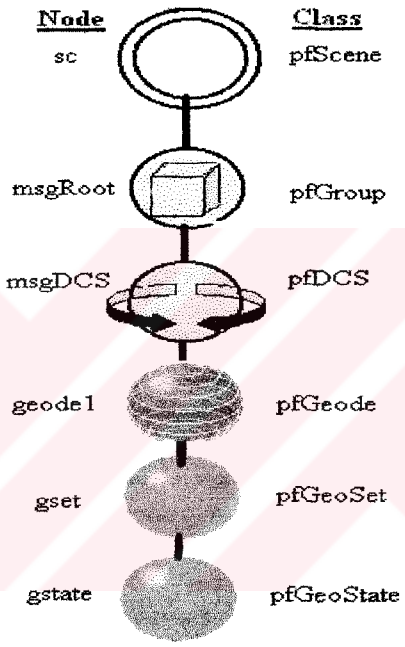


Figure 3.21 Message Center Scene Graph

The `gstate` node specifies the graphics state about the shape and texture of the object, which is created in the `gset` node. This node defines the geometry of rectangular tablets. The `geode1` node attaches the geometry and its appearance. The `MsgDCS` node is used to move the tablets through Y-axis. Thus it gives the ability to OOD to switch the receive and send modes.

3.3 ENVIRONMENT FEDERATE (EnviFd)

Environment Federate is designed to generate data related to virtual environment for other federates in order to render the scene. This federate is also designed to handle GIS server-federation communications for further studies.

When environment federate joined to federation, it waits for the task order message sent by Officer Scheduling the Exercise federate (OSEFd). The task order message contains the scenario, scenario location, the location limits, and the participants. Environment federate filters the name of the location, which scenario takes place and distributes it to the other federates. This location name is the name of the ocean/environment file, which is located in the file system of local host of each federate. Then federates load this file into their scene graphs to render. The ocean file used for illustrative purpose is the northeast part of HAWAII islands. The original file is in *Digital Elevation Data (DED)* format.

Environment federate also gives the ability for instructors

- To create some environmental effects such as fog at run time,
- To switch the time of day (e.g. from DAY to EVENING) at run time.

3.4 FEDERATION MANAGEMENT PROCESSES

Detailed design and implementation of *Federation Monitor Federate (FEDMONFd)* and *Officer Scheduling the Exercise (OSEFd)* is the subject of another thesis in progress and will be only sketched.

3.4.1 OSE FEDERATE (OSEFd)

Scenario Monitor simulates the Officer Scheduling the Exercise (OSE), therefore it is called OSE federate (OSEFd). Figure 3.22 depicts the modular structure of OSEFd. *Scenario Chooser Module* selects a scenario from *Scenario Database. Casting Module* distributes the scenario and the roles to the other simulation entities in the distributed environment and *Data Collector Module* accomplishes the collection of *Critical Performance Data (CPD)* by filtering.

Federate-initiated services are invoked on an instance of *RTIambassador.* The abstract class *Scenario Manager Ambassador* implements the callback functions of RTI by providing methods that the RTI calls during the OSEFd's tick period. During the tick period where the RTI is making calls to the OSEFd's Ambassador, the OSEFd directs updates and interactions to its database, queues the Ids of newly discovered objects and the objects that have left the federation.

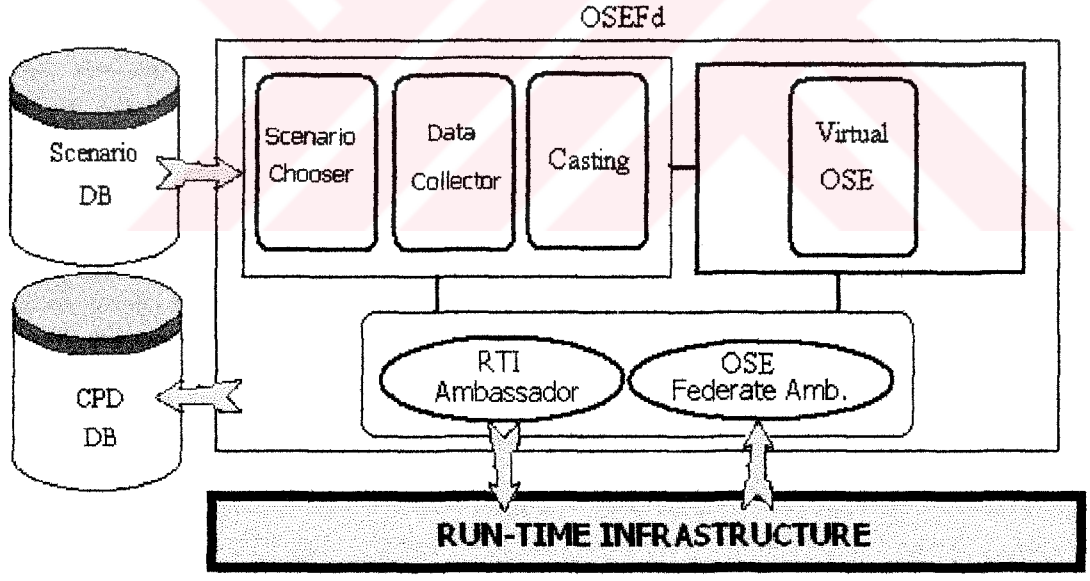


Figure 3.22 OSEFd Modular Structure

Data Collector Module design has been aimed at rapidly providing a baseline capability for logging a federation. Data Collector collects critical performance data for analysis of training.

3.4.2 FEDERATION MONITOR FEDERATE (FEDMONFd)

FEDMONFd mainly monitors the network and collects network-related data. It uses all the functionality of Management Object Model (MOM), which identifies objects and interactions used to manage a federation.

Rationale for Federation Monitor Federate:

- To analyze the performance of dead reckoning algorithms. Dead reckoning algorithms have some critical parameters, such as threshold and heartbeat rate, which should be fine-tuned to reduce overhead without significant loss in precision.
- To identify the bottlenecks of the system, by monitoring the network load, network delay, lost messages etc.

CHAPTER 4

HARDWARE ARCHITECTURE AND NETWORK ISSUES

This chapter covers the experimental hardware configuration, network issues, and some technical data for NSTMSS federation and the federates. This chapter also includes the Federation Execution Planner's Workbook (FEPW) for the sake of compatibility for HLA.

4.1 EXPERIMENTAL HARDWARE CONFIGURATION

Figure 4.1 graphically depicts the experimental system configuration, which is minimal, and describes the specific type of platforms for each of the federates in the system. Bracketed items indicate alternative components that could be hosted on a particular platform due to various execution considerations.

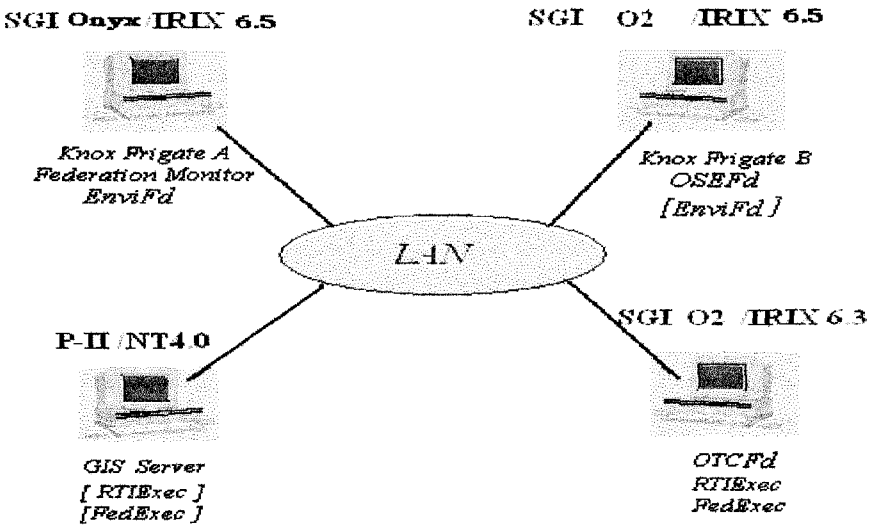


Figure 4.1 Execution Environment

4.2 TECHNICAL ENVIRONMENT

Specifying and recording the technical environment, where the federation execution takes place, is important because characteristics of the hardware and network plays a big role in efficiency of the federation execution. It is also important to specify the software and hardware requirements of the federates till these requirements define whether the federate runs or not.

Technical environment for NSTMSS federates are depicted in the Table 4.1. This table summarizes the hardware and software used by federates in the experimental federation executions.

Table 4.1 Technical Environment for NSTMSS Federates

Federates	**KNOXFd A**	**KNOXFd B**	**OTCFd**	**EnviFd**
HARDWARE	SGI Onyx RISK 12000 at 300MHz 1024MB RAM	SGI O2 RISK 12000 at 300MHz 640MB RAM	SGI O2 RISK 4400 at 200MHz 64MB RAM	Pentium-III 440 MHz 256 MB RAM
SOFTWARE	IRIX 6.5 Performer 2.2 IRIX C++ 7.3 RTI 1.3 v6	IRIX 6.5 Performer 2.2 IRIX C++ 7.3 RTI 1.3 v6	IRIX 6.3 Performer 2.0.2 IRIX C++ 7.1 RTI 1.3 v6	Windows NT4.0 Workstation Visual C++ 6.0 RTI 1.3v6
TIME MNGT.	Real Time	Real Time	Real Time	Receive Order
COMMENTS	FEDEX Running	-	-	RtiExec Running

4.3 FEDERATION EXECUTION PLANNER'S WORKBOOK

Besides specifying the technical environment, recording and specifying the characteristics of past federation executions can be used to validate expected federate performance, and to provide guidance for new federates. High Level Architecture provides a method for recording data in the federation execution: *The Federation Execution Planner's Workbook (FEPW)* is a mechanism designed to assist in the planning of HLA federation executions [DoD98d].

Federation Execution Summary Table identifies the federation execution and the RTI software used.

Table 4.2 Federation Execution Summary

Federation Execution Name		NSTMSS	Date of execution	11/01/99
Application Domain		Analysis	Version of Fed Ex	1
Purpose		Navy Sim	Sponsor	MASLAB
Concurrent Federation Executions	Number	1	Point of contact	Okan Topcu Liet.
	Names			MASLAB
				otopcu@nps.navy.edu.tr
RTI Software Used	Vendor	DMSO		
	Version	1.3 r 6	FEPW last mod date	08-Ara-99

Federate Summary Table summarizes characteristics of federates in the NSTMSS federation execution. This data identifies the federate and begins to add detail regarding how they operate in the federation execution.

Table 4.3 Federate Summary in Execution

Fed	Name	API	Size (MB)	Tick			Time Management			Host	LAN
				Rate (/sec)	Min Value	Max Value	Regulating (y or n)	Constrained (y or n)	Lookahead		
1	KNOXFdA	C++	20	10	0.001	0.005	N	N	0.00	1	1
2	KNOXFdB	C++	20	10	0.001	0.005	N	N	0.00	2	1
3	OTCFd	C++	16	10	0.001	0.005	N	N	0.00	3	1
3	EnviFd	C++	2	10	0.001	0.005	N	N	0.00	4	1

Host Table describes the computers used by the federation execution.

Table 4.4 Information About Hosts in Execution

Host	Identification				Processor			Operating System		Memory(MB)		%CPU
	Name	LAN	Network Address	Location	Vendor/ Model	#CPUs	Speed (MHz)	Name	Version	Total (MB)	Available (MB)	Available to FedEx
1	Samsun	1	144.122.216.54	MODSIM	SGI Onyx	1	300	IRIX	6.5	1024	1000	NA
2	Sves	1	144.122.216.55	MODSIM	SGI O2	2	300	IRIX	6.5	640	120	NA
3	Virtue	1	144.122.216.40	MODSIM	SGI O2	1	200	IRIX	6.3	64	30	NA
4	Sakarya	1	144.122.216.58	MODSIM	Pentium	1	450	NT	4.0	256	210	NA

LAN Description Table describes the network that is used by the federation execution.

Table 4.5 LAN Description

LAN	Physical Type	Bandwidth	
	(Ethernet, ATM,etc.)	Specified	Available
1	Ethernet	100 Mbit	100 Mbit

RTI Services Table describes the RTI services used by the federation execution. RTI service tables for federates are presented in the tables 4.6 through 4.8.

Object/Interaction Tables, Cross Reference Tables, and Data Representation Tables are the same with the Object Model Templates (OMT) documented in Appendix G.

Table 4.6 KNOXFd RTI Services

Federate	KNOX Federate v4.3

	Service	Used ?
Federation Mgt	Create Federation Execution	Y
	Destroy Federation Execution	Y
	Join Federation Execution	Y
	Resign Federation Execution	Y
	Register Federation Sync. Point	
	Confirm Sync. Point Registration	
	Announce Synchronization Point	
	Synchronization Point Achieved	
	Federation Synchronized	
	Request Federation Save	
	Initiate Federate Save	
	Federate Save Begun	
	Federate Save Complete	
	Federation Saved	
	Request Federation Restore	
	Confirm Federation Restoration Request	
	Federation Restore Begun	
	Initiate Federate Restore	
	Federate Restore Complete	
	Federation Restored	
Decl Mgt	Publish Object Class	Y
	Unpublish Object Class	Y
	Publish Interaction Class	Y
	Unpublish Interaction Class	Y
	Subscribe Object Class Attributes	Y
	Unsubscribe Object Class	Y
	Subscribe Interaction Class	Y
	Unsubscribe Interaction Class	Y
	Start Registration For Object Class	Y
	Stop Registration For Object Class	Y
	Turn Interactions On	Y
	Turn Interactions Off	Y
Object Mgt	Register Object Instance	Y
	Discover Object Instance	Y
	Update Attribute Values	Y
	Reflect Attribute Values	Y
	Send Interaction	Y
	Receive Interaction	Y
	Delete Object Instance	Y
	Remove Object Instance	Y
	Local Delete Object Instance	
	Change Attribute Transportation Type	
	Change Interaction Transportation Type	
	Attributes In Scope	
	Attributes Out Of Scope	
	Request Attribute Value Update	
	Provide Attribute Value Update	
	Turn Updates On For Object Instance	Y
	Turn Updates Off For Object Instance	Y

	Service	Used ?
Ownership Mgt	Unconditional Attribute Ownership Divestiture	
	Negotiated Attribute Ownership Divestiture	
	Request Attribute Ownership Assumption	
	Attribute Ownership Divestiture Notification	
	Attribute Ownership Acquisition Notification	
	Attribute Ownership Acquisition	
	Attribute Ownership Acquisition If Available	
	Attribute Ownership Unavailable	
	Request Attribute Ownership Release	
	Attribute Ownership Release Response	
	Cancel Negotiated Attr. Ownership Divestiture	
	Cancel Attribute Ownership Acquisition	
	Confirm Attribute Ownership Acq. Cancellation	
	Query Attribute Ownership	
	Inform Attribute Ownership	
	Is Attribute Owned By Federate	
Time Management	Enable Time Regulation	
	Time Regulation Enabled	
	Disable Time Regulation	
	Enable Time Constrained	
	Time Constrained Enabled	
	Disable Time Constrained	
	Time Advance Request	
	Time Advance Request Available	
	Next Event Request	
	Next Event Request Available	
	Flush Queue Request	
	Time Advance Grant	
	Enable Asynchronous Delivery	Y
	Disable Asynchronous Delivery	Y
	Query LBTS	
	Query Federate Time	
	Query Minimum Next Event Time	
	Modify Lookahead	
	Query Lookahead	
	Retract	
	Request Retraction	
	Change Attribute Order Type	
	Change Interaction Order Type	
Data Dist Mgt	Create Region	Y
	Modify Region	Y
	Delete Region	Y
	Register Object Instance With Region	
	Associate Region For Updates	
	Unassociate Region For Updates	
	Subscribe Object Class Attributes With Region	
	Unsubscribe Object Class With Region	
	Subscribe Interaction Class With Region	Y
	Unsubscribe Interaction Class With Region	Y
	Send Interaction With Region	Y
	Request Attribute Value Update With Region	
	Change Thresholds	
Support	Enable Class Relevance Advisory Switch	Y
	Disable Class Relevance Advisory Switch	Y
	Enable Attribute Relevance Advisory Switch	Y
	Disable Attribute Relevance Advisory Switch	Y
	Enable Attribute Scope Advisory Switch	
	Disable Attribute Scope Advisory Switch	
	Enable Interaction Relevance Advisory Switch	Y
	Disable Interaction Relevance Advisory Switch	Y

Table 4.7 OTCFd RTI Services

Federate	OTC Federate v3.0

	Service	Used ?
Federation Mgt	Create Federation Execution	Y
	Destroy Federation Execution	Y
	Join Federation Execution	Y
	Resign Federation Execution	Y
	Register Federation Sync. Point	
	Confirm Sync. Point Registration	
	Announce Synchronization Point	
	Synchronization Point Achieved	
	Federation Synchronized	
	Request Federation Save	
	Initiate Federate Save	
	Federate Save Begun	
	Federate Save Complete	
	Federation Saved	
	Request Federation Restore	
	Confirm Federation Restoration Request	
	Federation Restore Begun	
	Initiate Federate Restore	
	Federate Restore Complete	
	Federation Restored	
Decl Mgt	Publish Object Class	Y
	Unpublish Object Class	Y
	Publish Interaction Class	Y
	Unpublish Interaction Class	Y
	Subscribe Object Class Attributes	Y
	Unsubscribe Object Class	Y
	Subscribe Interaction Class	Y
	Unsubscribe Interaction Class	Y
	Start Registration For Object Class	Y
	Stop Registration For Object Class	Y
	Turn Interactions On	Y
	Turn Interactions Off	Y
Object Mgt	Register Object Instance	Y
	Discover Object Instance	Y
	Update Attribute Values	Y
	Reflect Attribute Values	Y
	Send Interaction	Y
	Receive Interaction	Y
	Delete Object Instance	Y
	Remove Object Instance	Y
	Local Delete Object Instance	
	Change Attribute Transportation Type	
	Change Interaction Transportation Type	
	Attributes In Scope	
	Attributes Out Of Scope	
	Request Attribute Value Update	
	Provide Attribute Value Update	
	Turn Updates On For Object Instance	Y
	Turn Updates Off For Object Instance	Y

	Service	Used ?
Ownership Mgt	Unconditional Attribute Ownership Divestiture	
	Negotiated Attribute Ownership Divestiture	
	Request Attribute Ownership Assumption	
	Attribute Ownership Divestiture Notification	
	Attribute Ownership Acquisition Notification	
	Attribute Ownership Acquisition	
	Attribute Ownership Acquisition If Available	
	Attribute Ownership Unavailable	
	Request Attribute Ownership Release	
	Attribute Ownership Release Response	
	Cancel Negotiated Attr. Ownership Divestiture	
	Cancel Attribute Ownership Acquisition	
	Confirm Attribute Ownership Acq. Cancellation	
	Query Attribute Ownership	
	Inform Attribute Ownership	
	Is Attribute Owned By Federate	
Time Mgt	Enable Time Regulation	
	Time Regulation Enabled	
	Disable Time Regulation	
	Enable Time Constrained	
	Time Constrained Enabled	
	Disable Time Constrained	
	Time Advance Request	
	Time Advance Request Available	
	Next Event Request	
	Next Event Request Available	
	Flush Queue Request	
	Time Advance Grant	
	Enable Asynchronous Delivery	Y
	Disable Asynchronous Delivery	Y
	Query LBTS	
	Query Federate Time	
	Query Minimum Next Event Time	
	Modify Lookahead	
	Query Lookahead	
	Retract	
	Request Retraction	
	Change Attribute Order Type	
	Change Interaction Order Type	
Data Dist Mgt	Create Region	Y
	Modify Region	Y
	Delete Region	Y
	Register Object Instance With Region	
	Associate Region For Updates	
	Unassociate Region For Updates	
	Subscribe Object Class Attributes With Region	
	Unsubscribe Object Class With Region	
	Subscribe Interaction Class With Region	Y
	Unsubscribe Interaction Class With Region	Y
	Send Interaction With Region	Y
	Request Attribute Value Update With Region	
	Change Thresholds	
Support	Enable Class Relevance Advisory Switch	Y
	Disable Class Relevance Advisory Switch	Y
	Enable Attribute Relevance Advisory Switch	Y
	Disable Attribute Relevance Advisory Switch	Y
	Enable Attribute Scope Advisory Switch	
	Disable Attribute Scope Advisory Switch	
	Enable Interaction Relevance Advisory Switch	Y
	Disable Interaction Relevance Advisory Switch	Y

T.C. YÜKSEKÖĞRETİM KURULU
DOKÜMANTASYON MERKEZİ

Table 4.8 EnviFd RTI Services

Federate	Environment Federate v1.1

	Service	Used ?
Federation Mgt	Create Federation Execution	Y
	Destroy Federation Execution	Y
	Join Federation Execution	Y
	Resign Federation Execution	Y
	Register Federation Sync. Point	
	Confirm Sync. Point Registration	
	Announce Synchronization Point	
	Synchronization Point Achieved	
	Federation Synchronized	
	Request Federation Save	
	Initiate Federate Save	
	Federate Save Begun	
	Federate Save Complete	
	Federation Saved	
	Request Federation Restore	
	Confirm Federation Restoration Request	
	Federation Restore Begun	
	Initiate Federate Restore	
	Federate Restore Complete	
	Federation Restored	
Decl Mgt	Publish Object Class	Y
	Unpublish Object Class	Y
	Publish Interaction Class	Y
	Unpublish Interaction Class	Y
	Subscribe Object Class Attributes	Y
	Unsubscribe Object Class	Y
	Subscribe Interaction Class	Y
	Unsubscribe Interaction Class	Y
	Start Registration For Object Class	Y
	Stop Registration For Object Class	Y
	Turn Interactions On	Y
	Turn Interactions Off	Y
Object Mgt	Register Object Instance	Y
	Discover Object Instance	Y
	Update Attribute Values	Y
	Reflect Attribute Values	Y
	Send Interaction	Y
	Receive Interaction	Y
	Delete Object Instance	Y
	Remove Object Instance	Y
	Local Delete Object Instance	
	Change Attribute Transportation Type	
	Change Interaction Transportation Type	
	Attributes In Scope	
	Attributes Out Of Scope	
	Request Attribute Value Update	
	Provide Attribute Value Update	
	Turn Updates On For Object Instance	Y
	Turn Updates Off For Object Instance	Y

	Service	Used ?
Own Mgt	Unconditional Attribute Ownership Divestiture	
	Negotiated Attribute Ownership Divestiture	
	Request Attribute Ownership Assumption	
	Attribute Ownership Divestiture Notification	
	Attribute Ownership Acquisition Notification	
	Attribute Ownership Acquisition	
	Attribute Ownership Acquisition If Available	
	Attribute Ownership Unavailable	
	Request Attribute Ownership Release	
	Attribute Ownership Release Response	
	Cancel Negotiated Attr. Ownership Divestiture	
	Cancel Attribute Ownership Acquisition	
	Confirm Attribute Ownership Acq. Cancellation	
	Query Attribute Ownership	
	Inform Attribute Ownership	
	Is Attribute Owned By Federate	
Time Mgt	Enable Time Regulation	
	Time Regulation Enabled	
	Disable Time Regulation	
	Enable Time Constrained	
	Time Constrained Enabled	
	Disable Time Constrained	
	Time Advance Request	
	Time Advance Request Available	
	Next Event Request	
	Next Event Request Available	
	Flush Queue Request	
	Time Advance Grant	
	Enable Asynchronous Delivery	Y
	Disable Asynchronous Delivery	Y
	Query LBTS	
	Query Federate Time	
	Query Minimum Next Event Time	
	Modify Lookahead	
	Query Lookahead	
	Retract	
	Request Retraction	
	Change Attribute Order Type	
	Change Interaction Order Type	
Data Dist Mgt	Create Region	Y
	Modify Region	Y
	Delete Region	Y
	Register Object Instance With Region	
	Associate Region For Updates	
	Unassociate Region For Updates	
	Subscribe Object Class Attributes With Region	
	Unsubscribe Object Class With Region	
	Subscribe Interaction Class With Region	Y
	Unsubscribe Interaction Class With Region	Y
	Send Interaction With Region	Y
	Request Attribute Value Update With Region	
	Change Thresholds	
Support	Enable Class Relevance Advisory Switch	Y
	Disable Class Relevance Advisory Switch	Y
	Enable Attribute Relevance Advisory Switch	Y
	Disable Attribute Relevance Advisory Switch	Y
	Enable Attribute Scope Advisory Switch	
	Disable Attribute Scope Advisory Switch	
	Enable Interaction Relevance Advisory Switch	Y
	Disable Interaction Relevance Advisory Switch	Y

CHAPTER 5

DEVELOPMENT METHODOLOGY

Analysis, design, and implementation of both federation and federates should be based on some methodologies. This chapter covers the methodology used in the course of the development of NSTMSS federation.

There are two issues in the development phase of a distributed simulation system; first analysis, design, and execution of a *federation* that is concerned with the interactions between federates. Second the design and development of *federates.* The two together describe a working simulation system.

The Federation Development Lifecycle comprises the design and implementation process for federation. Hence this work completely covers the HLA to gain experience; NSTMSS federation development lifecycle is based on *Federation Development and Execution Process (FEDEP),* which is proposed by HLA for the federation design and development.

The Federate Development Lifecycle is based on Object Oriented Analysis and Design (OOA&D) by following an Iterative and Incremental Lifecycle.

The following sections express why these methodologies are chosen, the details in using two different processes together, and the lessons learned.

5.1 FEDERATION DEVELOPMENT AND EXECUTION PROCESS (FEDEP)

Guidance in the process of developing HLA federations is provided by the Federation Development and Execution Process (FEDEP) [DMSO98c]. FEDEP identifies the key activities in the development of a federation, though it does not specify a single way of constructing an HLA federation. The FEDEP may be viewed as being more descriptive guidance than prescriptive instructions [SWRL98]. FEDEP is decomposed into five primary process areas:

- Requirements Definition
- Conceptual Model Development
- Federation Design
- Federation Integration and Test
- Federation Execution and Analysis

The figure 5.1 depicts the graphical representation of FEDEP.

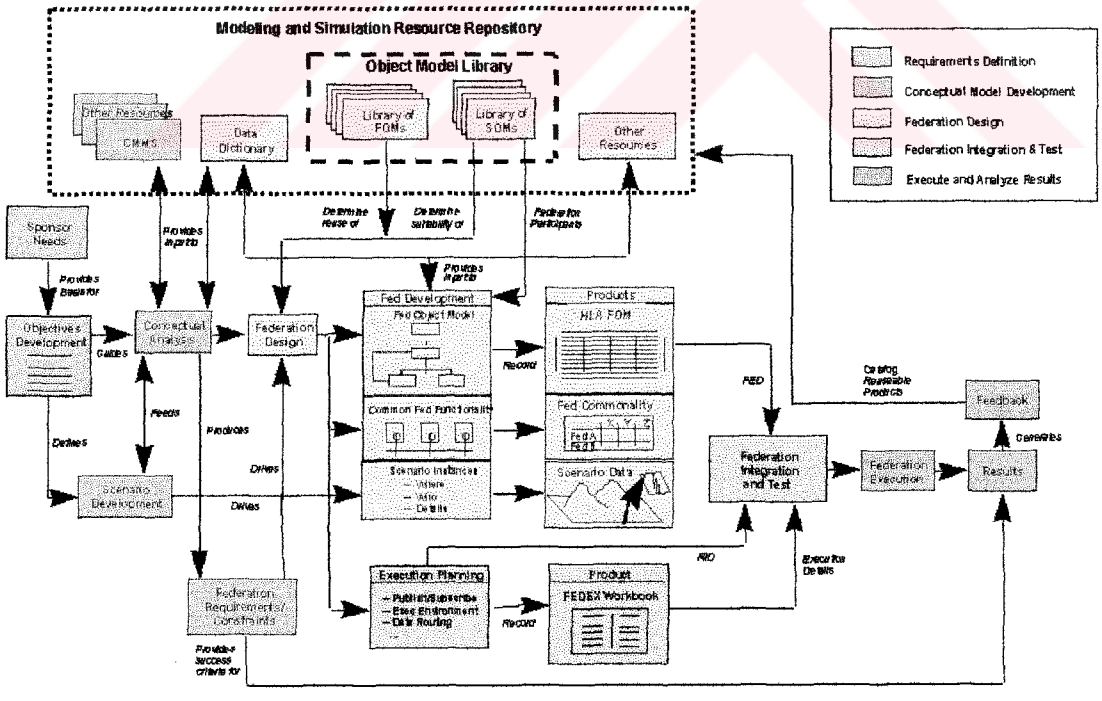


Figure 5.1 FEDEP Version 1.2 [DMSO98c]

It is based on 'Waterfall with Feedback Paradigm ', so it relies on a well-scoped sequence of phases.

In order to define the system requirements, a general problem statement should be formulated. Then system requirements analysis can be done. The system requirements also reflect the development or system objectives.

Conceptual model development consists of conceptual analysis and scenario development. A Scenario is strictly related to the general problem statement and it describes the system capabilities. Scenario is a step for conceptual analysis, which transforms objectives into functional characteristics and explains the interest domain. The two phases are strictly related with each other; therefore they are combined under the name "System Requirements Analysis", which is documented in Appendix A and the interest domain is explained in chapter 1.

Federation design and development is the first activity to define the system components to satisfy the requirements. At this stage, capabilities and interests of the federates are being defined, Federation Object Model (FOM) and Simulation Object Models (SOM) are constructed, and data routing properties are defined. Chapter 2 is dedicated to Federation design and development. At the end of this work, the Federation Execution Details (FED) file and RTI Initialization Data (RID) file are prepared.

Federation integration phase involves the execution environment and the Federation Execution Planner's Workbook (FEPW) is prepared.

In the federation execution phase, all federation participants are now operating as an integrated whole to generate the required measures of merit and thus achieve the stated federation objectives/requirements.

At conclusion section of this chapter, I present the development lifecycle in detail.

5.2 OBJECT ORIENTED ANALYSIS & DESIGN (OOA&D)

The FEDEP assumes that the SOMs of federates are at hand and ready to be reused in the Modeling and Simulation Resource Repository. In our development this is not the case. While developing federation, it is necessary to analyze, to design, and to implement the federates that are defined in federation design stage.

HLA supports an object-oriented approach in a whole. Although there are big differences between HLA and OOAD principles and concepts, the object-oriented approach helps easily formalize the static and dynamic properties of the simulation system. Therefore, object-oriented approach was used in the analysis and design phases of the federates. The analysis and design notation used is Unified Modeling Language (UML v1.1). The object-oriented programming is the last link of the object-oriented chain, after analysis and design. ANSI C++ object-oriented language is chosen, because both RTI and Performer API are available in C++.

UML is a language for representing object-oriented models, but the process for the creation of these models is not described by UML. Therefore, the OOAD should be based on a methodology. We have employed the iterative and Incremental Lifecycle Methodology.

The fact that each federate works correctly by itself does not imply that it will work properly together with the others in a federation. The federation design and integration should be done parallel with the design and implementation of federates. The iterative and incremental lifecycle should be used with

FEDEP in some way. The conclusion and assessment section of this chapter presents "the way" we took in development of NSTMSS.

5.3 CONCLUSIONS AND ASSESSMENT

DEVELOPMENT LIFECYCLE:

In the development lifecycle of NSTMSS, we have combined the FEDEP and iterative and incremental lifecycles. The figure 5.2 shows a single iteration of the lifecycle. Number of runs are project dependent. The first run takes much time than other runs and also has least functionality with stubs and processes. Each run enriches an existing program by adding functionality. For example, in our first iteration OTC federate existed as a stub, and it was completed in the next iterations.

Requirements analysis phase and conceptual model development phase include:

- Defining the interest domain
- Defining the problem statement
- Defining objectives
- Defining system requirements
- Defining GUI requirements
- Defining HW requirements
- Defining scenarios

Federation design and development phase and analysis and design of federates are parallel activities. Formation design phase includes:

- Defining system components
- Defining initialization and termination scenarios of the system
- Defining system wide principles
- Defining federation components
- Defining problem domain entities in terms of HLA objects and interactions
- Constructing FOM

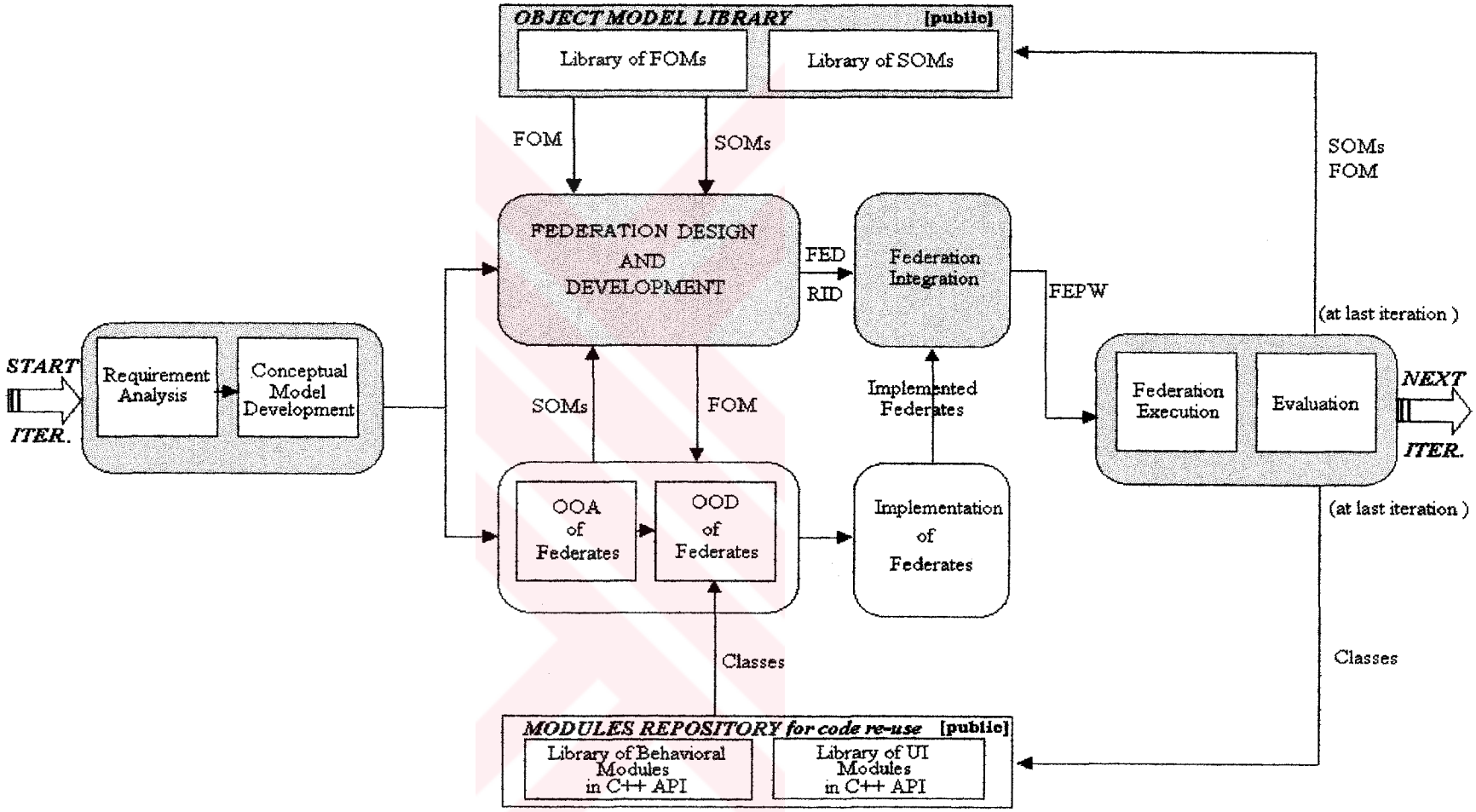


Figure 5.2 Development Lifecycle of NSTMSS

OOA&D phase includes:

- Mapping all requirements into use cases for each federate
- Defining interactions in terms of sequences and collaborations
- Defining classes from use cases and interactions
- Designing GUI
- Defining the associations between classes
- Constructing SOMs

Federation Integration and implementation of federates are also parallel phases. Federation execution is the test phase.

Object Model Library (OML) and Modules Repository are public libraries. OML is for design reuse, while modules repository is for code-reuse.

EXPERIENCE IN UML:

A DIS application, as its name implies, have some significant characteristics: It is distributed, which means that many players can participate into the virtual environment; it is interactive, which simulation can be dynamically guided by the human operators; and it is a 'simulation', which models the real world and/or real world entities.

It has been expected that all the functionality of the system should be the same with real world in terms of user interfaces, behavioral models and polygonal models. The analysis and design of such an application must reflect the boundaries of the system, must model the behaviors of domain objects, must represent static and dynamic structure of the system, and must reveal the physical implementation of the system. Especially, the analysis phase must reflect the behaviors of domain objects, all the functional and non-functional requirements for the system.

In this regard, UML presents all the diagram types to depict such a system in different views. UML provides a good solution for capturing, documenting, visualizing, and constructing all the artifacts of a DIS application.

CHAPTER 6

DISCUSSION AND CONCLUSIONS

Michael Zyda and his colleagues state: "Collaborative virtual environment research combines graphics, networking, human perception, and distributed computing issues. However, these facts betray a disappointing lack of coordination. Most such projects from graphics researchers consider the network a magical transport mechanism, while network scientists treat graphics content as no different than any other [CWZ99]."

NSTMSS is a case study, which shows the way how the different tools and architectures can be combined to achieve a functional distributed interactive simulation.

This chapter summarizes the contributions of this research and presents the objectives that are achieved.

6.1 CONTRIBUTIONS OF THIS WORK

The principal goal of this research was to gain experience with the HLA, while constructing a testbed for DIS studies.

- The design and implementation of a simple shiphandling simulator is accomplished.

- The design and implementation of a plan-view tactical level simulation is accomplished.

- Platform level and tactical level simulations are linked in order to create a multi-level, interactive, and distributed training simulation system.

- A reusable object model library composed of FOM and SOMs is constructed by developing a NSTMSS FOM and KNOX federate, OTC federate, and Environment federate SOMs.

- A reusable behavioral modules repository and user interface modules repository are constructed in C++ API.

- FEDEP methodology improvement is achieved by
  - Devising a graphical federation design notation,
  - Practicing iterative and incremental development approach.

- NSTMSS is already serving as a testbed for the following research undertaken by several M.S. students:
  - "Using and testing HLA Management Object Model (MOM) services by developing federation management federates",
  - "Integration of navigation charts into virtual environments",
  - "The efficiency of multicast protocols over a NVE",
  - "Component-based architectural framework specification for HLA".

- Although the scope of this thesis does not include research in graphics, the design and implementation of a scene graph is done by using off-the-shelf graphics library: Performer.

NSTMSS software (currently in version 1.0) is publicly available and can be obtained from "http://virtue.ceng.metu.edu.tr/~nstmss". Practical matters and experiences about RTI, FEDEP, FED file, C++, and Rational Rose are presented in appendix E.

6.2 SUGGESTIONS FOR FUTURE WORK

TO IMPROVE REALIZM:

1. 'Ship Control Voice Commands' Interface

Bridge officers control the movement of their ships by uttering standard rudder and engine voice commands (explained at requirement analysis section) to personnel manning steering and engine controls. Speech recognition software should be used to "decipher/understand" the verbal commands and affect the appropriate behavior of the ship's hydrodynamic model. Considering the simple grammar and limited vocabulary, this should be feasible within the current technology.

2. Tactical Voice Message Interface

Tactical communication between ships is in form of voice communication in UHF channels. An interface that will enable a voice communication between federates should be developed. Some commercial solutions exist and these can be integrated to NSTMSS.

3. Air Radar Modeling

Air radar model and its interface will enable system to expand towards a joint operation by the participation of air-warfare models.

4. Surface Radar Modeling and Interface

Surface radar model and its interface will import the combat-abilities to ships.

5. Navigation Lights

Navigation lights has an importance in sea collision avoidance and it should be adapted to the system to increase the training capabilities.

6. Navigation Sound Communications

Navigation sound communication has an important role in avoiding sea collisions.

7. Flags and Semaphore Communication Interface

Semaphore and flag communication are sometimes used in tactical maneuvering exercises instead of voice communication for security reasons. It will increase the training capabilities and reality.

8. Local Meteorology Station Federate

Although some atmospheric effects such as fog is added to the system, a broad range of these effects should be introduced to the system by a new federate.

9. Ocean Modeling

Modeling uncontrollable forces such as winds, currents, waves will affirmatively increase the correctness of ship movements.

10. Enhancements Over Coordinate System

It is important in joint simulations that the transformation between geodetic and geocentric coordinate systems should be based on World Geodetic System 1984 (WGS-84) for more accuracy and efficiency.

GENERAL SUGGESTIONS:

1. Scenario Scripting Language

It is essential that more generic and flexible scenarios mean more realistic training. To prepare generic scenarios easily by Scenario Management Staff, it is necessary to develop a scenario scripting language.

2. Unknown-Object Geometry Distribution

If, for example, a sailing ship is joined in the virtual environment, how will the federates that discover her, obtain her polygonal model? This question may be answered in many ways by distributed or centralized approaches. In a distributed virtual environment, efficient distribution of polygonal models of entities between participants is worth to research.

3. MOM Testing

RTI provides an interface called Management Object Model (MOM) to manage and monitor federation execution. The federation management federates may use MOM instead of developing more application-specific solutions. The MOM efficiency should be tested.

4. RTI Time Management Schemes Testing

RTI provides many time management schemes. The efficiency and interoperability between these schemes should be investigated.

5. Risk Management in FEDEP

Any human activity has its share of risks and federation development is no exception to this rule. A risk management scheme should be adapted to FEDEP. "*The Risk Management in Iterative Development*" approach may be suitable for the integration of our approach for federation development.

6. Development of a Notation for Federation Design As a UML Extension

OMT components represent information about classes of objects, their attributes and their interactions in a *tabular way*. The visualization of this data will provide more readability, and simplicity over complex and large-scale federation designs.

7. Conceptual Model of Mission Space (CMMS) Development

HLA does not specify enough details to ensure interoperability of simulations. That is acceptable because it has to cover a broad range of issues. However, to remedy the situation a *Conceptual Model of Mission Space (CMMS)* based on the needs of Turkish Armed Forces should be developed as a first step.

8. Area of Interest Management and Multicast Transport Protocols

RTI uses a multicast protocol to handle many data distribution management. But, there exist many multicast protocols. A study should be carried out to determine the *efficient* multicast protocols, which are suitable with RTI.

9. Intelligent Agents in Virtual Environment

An STG can be composed of ships driven by virtual OODs as well as human OODs. A virtual OOD can be realized as an intelligent agent, a piece of software that acts autonomously to achieve a given goal.

APPENDIX – A

SYSTEM REQUIREMENTS ANALYSIS

The requirements phase captures the functional and non-functional requirements for the system. It presents an external or "black box" view of system behavior from the end user's perspective. For each process that is to be automated, I identify, categorize, and prioritize high-level requirements. For each functional requirement, I identified one or more use cases and documented each use case by using the *textual use case specification template*.

I also document the domain information needed for further analysis and design.

A.1 SOFTWARE REQUIREMENTS

A.1.1GENERAL REQUIREMENTS:

1. Type of ships that must be modeled: KNOX class frigate.

   Existing KNOX Class Frigates of the Turkish Navy are as follows:

   TCG MUAVENET (F-250)

   TCG ADATEPE (F-251)

   TCG KOCATEPE (F-252)

   TCG ZAFER (F-253)

   TCG TRAKYA (F-254)

   TCG KARADENIZ (F-255)

   TCG EGE (F-256)

   TCG AKDENIZ (F-257)

2. KNOX Class Frigates main characteristics

General Characteristics:

Length: 133.5m

Width: 14.3m

Draft: 8m

Deplacement: 3011tons (standard), 4260t(full)

Max speed: 27kts

Personnel: 20 officers, 268 enlisted

Artillery Characteristics (not used):

Guns: 1*127 mm/54(5in); 1*Mk15 Phalanx (CIWS)

Missiles: 8*Harpoon SSM; ASROC Mk16 launcher

Torpedoes: 2*324mmMk32 twin launcher for Mk46s

Machinery Characteristics:

Propulsion: 2 non-pressure combustion engineering

Shafts: 1

Horsepower: 35000hp

Range: 4000nm@20kts

Others (not used):

Radars: SPS40B (Air Search), SPS10, SPS-67 (Surface Search), Marconi LN66 (Navigation), SPG-53A (Fire Control)

Sonars: SQS-26CX; SQR-18A (v) 1

Helicopters: 1*AB212 ASW

Turning Dimensions:

Since NSTMSS is distributed openly, all of the source code is unclassified. Therefore rotation radius must be unclassified and not given here.

Table A.1 Turning Dimensions

Speed (kts)	*Rudder Angle (degree)*	*Rotation Radius (yds)*
5	5	N/A
	10	N/A
	15	N/A
	20	N/A
	25	N/A
10	5	N/A
	10	N/A
	15	N/A
	20	N/A
	25	N/A
15	5	N/A
	10	N/A
	15	N/A
	20	N/A
	25	N/A
20	5	N/A
	10	N/A
	15	N/A
	20	N/A
	25	N/A
25	5	N/A
	10	N/A
	15	N/A
	20	N/A
	25	N/A

2. Types of Formations that must be modeled:

Bearing Formations (True, Relative)

A.1.2 STG REQUIREMENTS:

1. STG must consist of at least 2 surface ships.
2. One STGC must be assigned to per STG by OSE.
3. A three digit unique number must be assigned to per STG by OSE (e.g. STG 100)

A.1.3 TACTICAL MESSAGE SYSTEM (TMS) REQUIREMENTS:

Tactical Message System describes the message formats, defines the senders and the receivers. All messages must be handled via TMS. It is consisted of weather reports, Task Orders, and Formation Messages.

1. **Weather Report:**

   Type: Broadcast/On-Demand

   Format:

   Fm: [Local Meteorology Station]

   To: [All Ships || Demanding Ship]

   Subject: Weather Report

   BR (Notation indicates that a 'break' separating the header of the message from its body)

   Validate From ----- To ------(duration 12 hours)

   *Visibility: ---------**Nm (nautical mile)***

   Sea State:------ (1.Calm,2.Smooth,3.Slight,4.Moderate,5.Rough, 6.Very Rough, 7.High, 8.Very High, 9.Phenomenal)

   Wind Course: ---/----(name of the wind)(000 N, 045 NE, 090 E, 135 SE, 180 S, 225 SW, 270 W, 315 NW)

   *Wind Speed: ---- **Kts (Knots)***

Example:

Fm: Bandirma Meteorology Station

To: TCG TRAKYA

Subject: Weather Report

BR

Validate From 0001 to 1200

Visibility: 20 nm

Sea State: 1

Wind Course: 045-NE

Wind Speed: 12 kts

2. *Task Order (TASKORD)* message:

General Format:

Fm: [OSE]

To: [STGC]

Cc: [If required]

Subject: TASKORD

BR

Body:

Example;

Fm: OSE

To: STGC100

Cc: STG100

BR

1.Cpt.Okan Topcu is assigned as STGC100

2.STG100 is composed of

TCG MUAVENET, TCG ADATEPE, TCG TRAKYA.

3. a. All ships are in the area of anchor at Golcuk

b. STG will departure from Golcuk at 1330 Saturday and they will perform formation exercises in Marmara Sea, and they will arrive Canakkale Harbor at 0800 Sunday.

c. STGC will inform the position of STG per 8 hours to OSE.STGC will also report the departure and arrival times of STG to OSE.

3. Formation Messages:

General Format:

Fm: [STGC]||The Ship

To: [STG]||STGC

Cc: [If Required]

BR

Body:

Formation Order Message Example:

Fm: STGC100

To: STG100

BR

Formation Id: 090 True

Guide ship: TCG TRAKYA

Sequence No: 1 TCG TRAKYA, 2 TCG MUAVENET, and 3 TCG KARADENIZ

Formation Course: 180

Formation speed: 12kts

Interval: 1000 yds

Formation Turn / S-Turn Message Example:

Fm: STGC100

To: STG100

Cc: TCG Muavenet

BR

Formation Course: 200

Formation Speed: 15kts

Interval: 1500 yds

Formation Feedback Message Example:

Fm: TCG TRAKYA

To: STGC100

Cc: STG100

BR

TCG TRAKYA has taken position.

4. Ship Control Commands:

a. Helmsman Commands

- ❖ Right/Left (R/L) full rudder
- ❖ R/L 15 degrees rudder
- ❖ Hard R/L rudder
- ❖ Rudder Amidships
- ❖ R/L to course 050 True

b. Engine Order Telegraph (EOT) Commands

Table A.1-2 Engine Order Telegraph Commands and RPM Values

All Engines/Right Engine/Left Engine	Ahead Flank	675
	Ahead Full	625
	Ahead Standard	450
	Ahead 2/3	250
	Ahead 1/3	100
	Stop	0
	Back 1/3	75
	Back 2/3	175
	Back Full	250

A.1.4 OOD REQUIREMENTS:

1. OOD must see the *ahead sector*, right & left sector, *and astern sector* of his/her ship. Figure A.1 gives an illustration of ahead sector view of the ship.

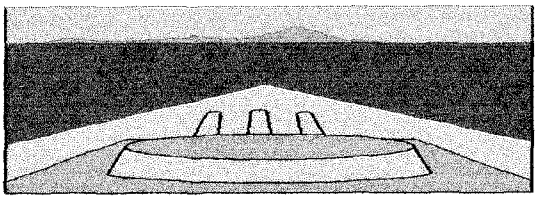


Figure A.1 Ahead Sector view of the ship

2. OOD must see and use *The Navigation Radar.*

3. OOD must see and use *The Helm.*
4. OOD must see and use *The Engine Order Telegraph.*
5. OOD must see and issue *Messages.*
    a. Received Messages:
        i. TASKORD
        ii. Weather Report
        iii. Formation Messages (Formation Order Message, Formation Turn Message, Formation S-Turn Message)
    b. Sent Messages:
        i. Formation Messages (Formation Feedback Message)
6. OOD must see the Global Positioning System (GPS) Display.
7. OOD must see the *Navigation Charts* of the area.
8. Single User Interface/OOD Graphical User Interface (GUI) Requirements
    a. Command Toolbar Interface (CTI):
        i. 3D views (Right, left, ahead and back sectors of the ship)
        ii. Navigation Map
        iii. Radar Scope
        iv. Ship Control Commands
        v. Message Center and Message-In Indicator
        vi. GPS Display

A.1.5 OTC REQUIREMENTS:

1. OTC must see the *Tactical View* (Plan-View) of *Operational Area.*
2. OTC can move to another ship in the STG (optional).
3. OTC must send feedback to OSE regularly (coordinates of STG) or spontaneously (emergency events, enemy contacts) depending on the situation.

A.1.6 OSE/SCENARIO MONITOR REQUIREMENTS:

1. OSE must initialize the exercise environment. Initialization should include the number of the ships included in the STG.
2. OTC must send TASKORD message to all participants.

3. OSE may produce some emergency situations:
    a. Man-overboard
    b. Malfunctioning of rudder
    c. Breakdown of engines
4. OSE must choose a scenario from 'scenario database'.
5. OSE may change weather conditions.
6. OSE must collect Critical Performance Data (CPD) for after-action review.

A.1.7 FEDERATION MONITOR (FEDMON) REQUIREMENTS:

The aim of FedMon is to be able to determine the performance bottlenecks in different situations and thus help fine-tune the system so that the implemented system will be highly scalable.

1. Dead Reckoning Message Monitoring
2. Time Management Message Monitoring

A.1.8 NAVIGATION RADAR REQUIREMENTS:

1.Marconi LN66 Navigation Radar General Characteristics:
    a. Max. Range
    b. Min. Range
    c. Focused Range
    d. 10 inch Display Unit
2. OOD must be able to change scale between min. and max. ranges
3. GUI Requirements:
    a. Scope of radar (display unit)
    b. Bearing button
    c. Bearing display
    d. Distance button
    e. Distance display
    f. Focused Range Button

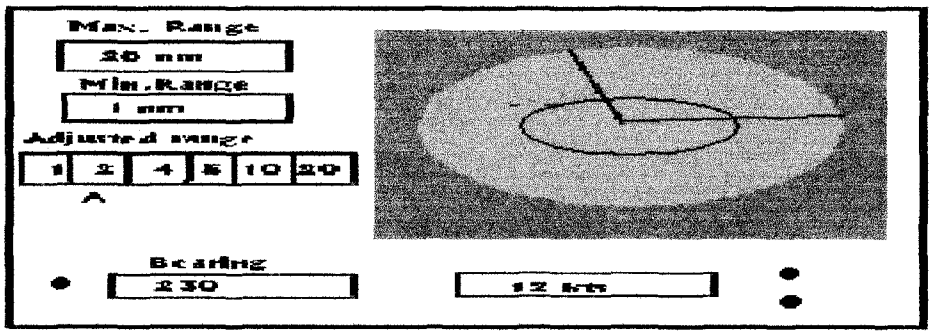


Figure A.2 Radar GUI

A.1.9 HELM REQUIREMENTS:

1.GUI Requirements:

a. Gyro Compass Display

b. Rudder Indicator (Right 5,10,15,20,25 –Left 5,10,15,20,25)

c. Rudder Control Joystick

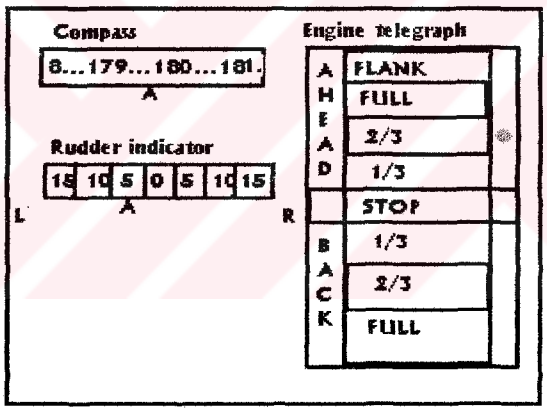


Figure A.3 Compass, Rudder Indicator and Engine Telegraph

A.1.10 ENGINE ORDER TELEGRAPH REQUIREMENTS:

1.GUI Requirements (See Figure A.3):

a. Engine Status Display

b. Engine Control Button

A.1.11 GPS DISPLAY REQUIREMENTS:

1. GPS must give the position of the ship as Latitude (LAT) and Longitude (LONG)

2. GPS must give the current speed in knots.
3. GUI Requirements:
   a. GPS display

A.1.12 NAUTICAL GEOGRAPHIC INFORMATION SYSTEMS (GIS) REQUIREMENTS:

1. Electronic Chart Formats:
   a. ENC IHO/S57 version3
   b. Digest Digital Nautical Chart (DNC)/ Vmap
   c. Raster Charts (e.g. Admiralty Raster Chart System ARCS)
2. Functions to allow OTC to manipulate plan-view:
   a. Zoom in
   b. Zoom out
   c. Scroll displayed area
   d. Change scale
3. Functions to support 3D views (right left and ahead sectors of the ships):
   a. Depth and height data of the given position
   b. Navigation aids (Lights, Buoys)
4. Functions to support navigation map view:
   a. Zoom in
   b. Zoom out
   c. Change scale
   d. Scroll displayed area
   e. Rotate-optional
   f. Drawing routes-optional
5. Functions to support Navigation Radar:
   a. Change Scale

A.1.13 NAUTICAL CHART REQUIREMENTS:

1. OOD must be able to select the nautical/marine chart of the region.
2. Map scale can be changed by the OOD.

3. GUI Requirements:

a. Scale Button

b. Chart Selection Button

c. Route Drawing Toolbar

A.1.14 FORMATION REQUIREMENTS:

1.Formation Formats:

a. Bearing Formations:

i. Line of True Bearing Formations:

The guide ship must immediately turn to the ordered formation course and adjust its speed into ordered formation speed. Other ships must maneuver to take her position, which are the true angle and the ordered sequence number. When a ship takes her position, she must report to OTC and other ships in the formation.

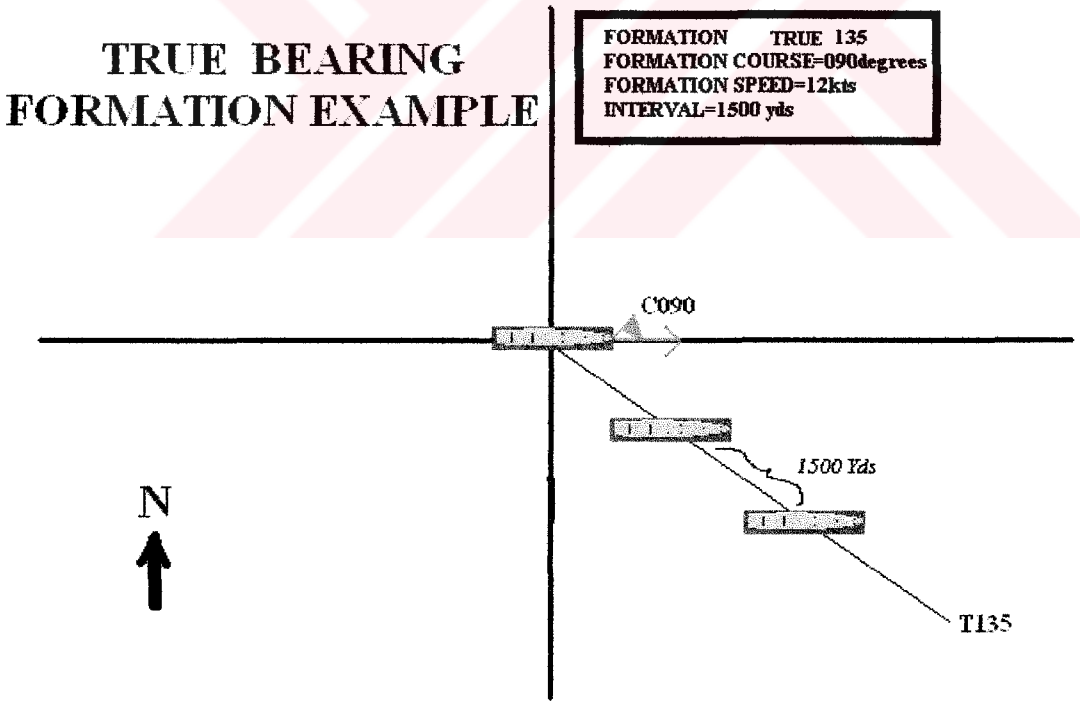


Figure A.4 True Bearing Formation Example

ii. Line of Relative Bearing Formations:

The guide ship must immediately turn to the ordered formation course and adjust its speed into ordered formation speed. Other ships must maneuver to take the position, which is defined as the relative angle with respect to the guide and the sequence number. When a ship takes her position, she must report it OTC and other ships in the formation.

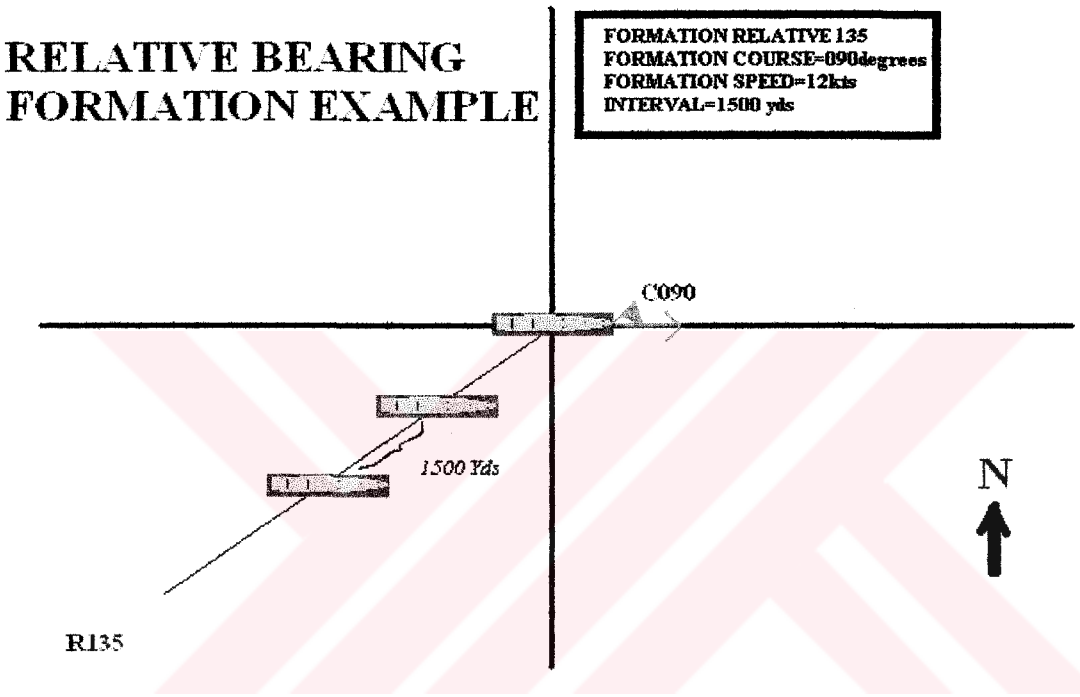


Figure A.5 Relative Bearing Formation Example

b. Turn:

All the ships in the formation must turn from the turning point of the guide ship. This is illustrated in the figure A.6:

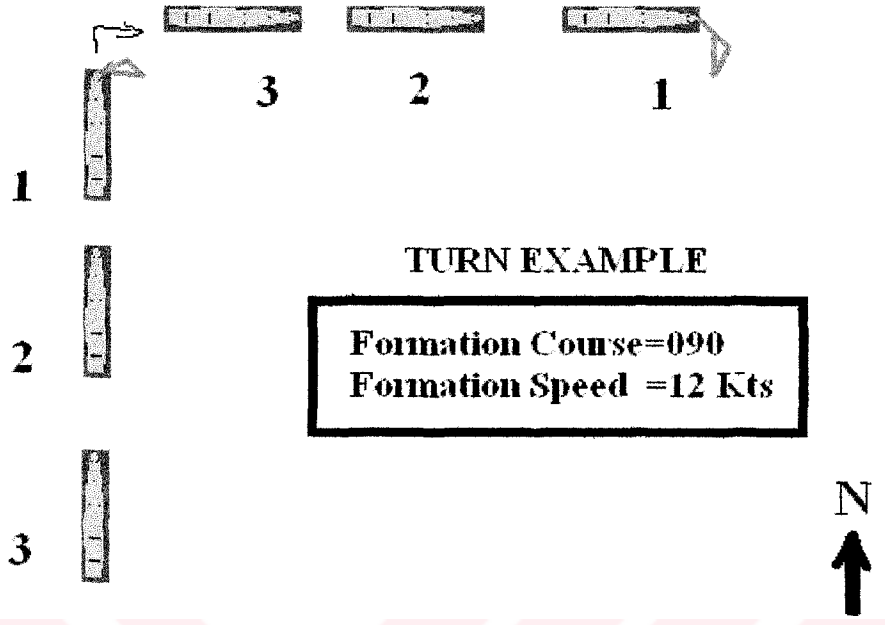


Figure A.6 Turn Example

c. S-Turn (Simultaneous Turn):

All the ships in the formation must immediately turn to the ordered formation course. This is illustrated in the figure A.7:

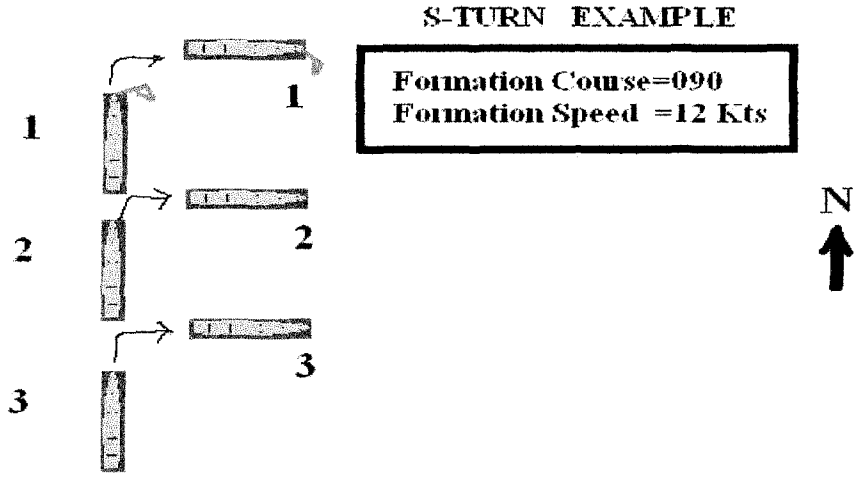


Figure A.7 S-Turn Example

2. Rules Of the Road (International Regulations for Preventing Collisions at Sea):

a. Rule 13-(b):

A vessel shall be deemed to be overtaking when coming up with another vessel from a direction more than 22.5 degrees abaft her beam, that is, in such a position with reference to the vessel she is overtaking that at night she would be able to see only the stern light of that vessel but neither of her sidelights [USCG95].

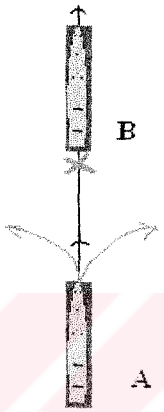


Figure A.8 Rule 13-(b) Example

b. Rule 14 -(a):

When two power-driven vessels are meeting on reciprocal or nearly reciprocal courses so as to involve risk of collision each shall alter her course to starboard so that each shall pass on the port side of the other [USCG95].

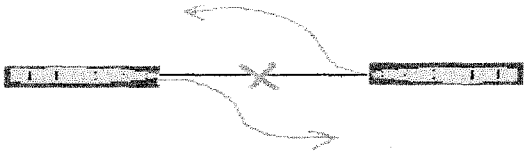


Figure A.9 Rule 14(a) Example

c. Rule 15:

When two power-driven vessels are crossing so as to involve risk of collision, the vessel which has the other on her own starboard side

shall keep out of the way and shall, if the circumstances of the case admit, avoid crossing ahead of the other vessel.

d. Rule 17-(a)(b)(c):

(a) (i) Where one of two vessels is to keep out of the way, the other shall keep her course and speed.

(ii) The latter vessel may, however, take action to avoid collision by her maneuver alone, as soon as it becomes apparent to her that the vessel required to keep out of the way is not taking appropriate action in compliance with these Rules.

(b) When, from any cause, the vessel required to keep her course and speed finds herself so close that collision cannot be avoided by the action of the give-way vessel alone, she shall take such action as will best aid to avoid collision.

(c) A power-driven vessel which takes action in a crossing situation in accordance with subparagraph (a)(ii) of this Rule to avoid collision with another power-driven vessel shall, if the circumstances of the case admit, not alter course to port for a vessel on her own port side.[USCG95]

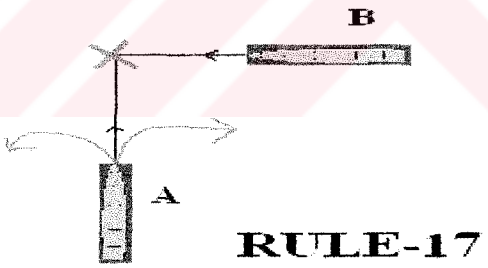


Figure A.10 Rule 17(a)(b)(c) Example

d. Rule 18-(a):

(a) A power-driven vessel underway shall keep out of the way of:

(i) a vessel not under command;

(ii) a vessel restricted in her ability to maneuver;

(iii) a vessel engaged in fishing; and

(iv) a sailing vessel [USCG95].

T.C. YÜKSEKÖĞRETİM KURULU
DOKÜMANTASYON MERKEZİ

A.2 HARDWARE REQUIREMENTS

A.2.1 GENERAL REQUIREMENTS:

All software must run on one of IRIX, NT, and Linux boxes.

APPENDIX – B

OBJECT ORIENTED ANALYSIS (OOA)

The primary goal of this phase is to develop a model of what the system will do (analysis) without regard for how it will be done (design). During the analysis phase, the focus is on domain objects and there is no regard for implementation details (e.g. collection classes, memory management, data types, etc.). I refine the textual use case specifications into use case diagrams and I construct a behavioral model for domain that exhibits significant dynamic behavior using state, collaboration and sequence diagrams. All notations are based on Unified Modeling Language (UML) v1.1.

B.1 SYSTEM USERS/ACTORS:

Analysis starts with the search for the actors (categories of users) of NSTMSS. An *Actor* represents a role played by someone or something interacting with the system [MM97]. It models a kind of object outside the domain of the system itself that interacts directly with the system. The name of the actors describes the role played by the user. They are divided up into the following categories:

- Officers Of the Deck (OODs) (minimum 2)
- Officer in Tactical Command (OTC) (min.1)
- Officer Scheduling Exercise (OSE) (min.1)
- Federation Monitor (min.1)

Here are their corresponding graphical UML representations (Figure B.1):

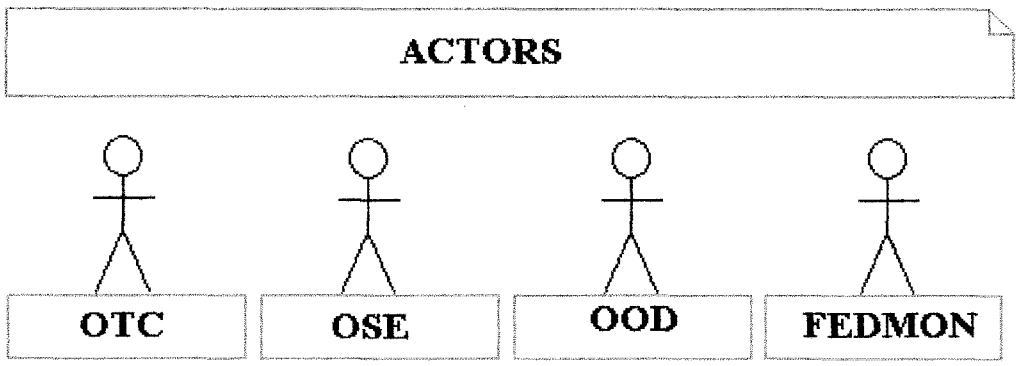


Figure B.1 System Users/Actors

OOD and OTC are *principal actors*, people who use the main system functions. OSE and FEDMON are *secondary actors*, people (in fact they are processes, which model the real users) that perform administration and monitoring tasks.

The *OOD* is the officer in responsible to control and maneuver ship movements while the ship is underway. He/she acts as a direct representative of the Commanding Officer (CO)/Captain.

OTC is the superior officer who is in tactical command of STG to achieve the given duty.

OSE is the authority, which plans all the surface action and determines the duty.

FEDMON is the researcher who wants to analyze the system efficiency.

B.2 USE CASE DIAGRAMS:

A use case diagram presents a collection of use cases and actors and is typically used to specify or characterize the functionality and behavior of a

whole application system interacting with one or more external actors. During analysis, the use case diagrams are used to capture the system requirements and understand what the system is supposed to do.

B.2.1 USE CASE IDENTIFICATION

The goal of the use case study is to determine what each actor expects from the system. They describe the behavior of the system from a user's standpoint by using actions and reactions. They allow the definition of the system's boundary, and the relationships between the system and the environment. Use cases complement the object-oriented approach and form the basis of the analysis and design process.

The functional requirements of the actors may be broken as follows in the Table B.1:

Table B.1 System Top Level Use Cases

ACTOR	MAIN USE CASES
OOD	Message handling
	Ship handling
OTC	Message handling
	STG handling
OSE	Scenario handling
	Monitoring
FEDMON	Monitoring

B.2.2 TOP LEVEL "OOD" USE CASE DIAGRAM

The following diagram (Figure B.2) represents the use cases for the OOD. To operate the ship in the STG, the OOD must handle both the ship movement and the tactical messages. Here, the main use cases: '*message handling*' and '*ship handling*' is being magnified to show more details. Message handling use case "uses" '*receive message*' and '*send message*' use cases. A "*uses*" generalization from use case A to use case B indicates

that an instance of A can perform all the behavior described by B. The behavior described by B is mandatory to use case A. Report to OTC use case is an extension of the *'send message'* use case. An "*extends*" generalization from use case A to use case B indicates that an instance of B can be extended to perform the entire behavior described in A. The behavior described by A is optional and triggered by a condition in use case B.

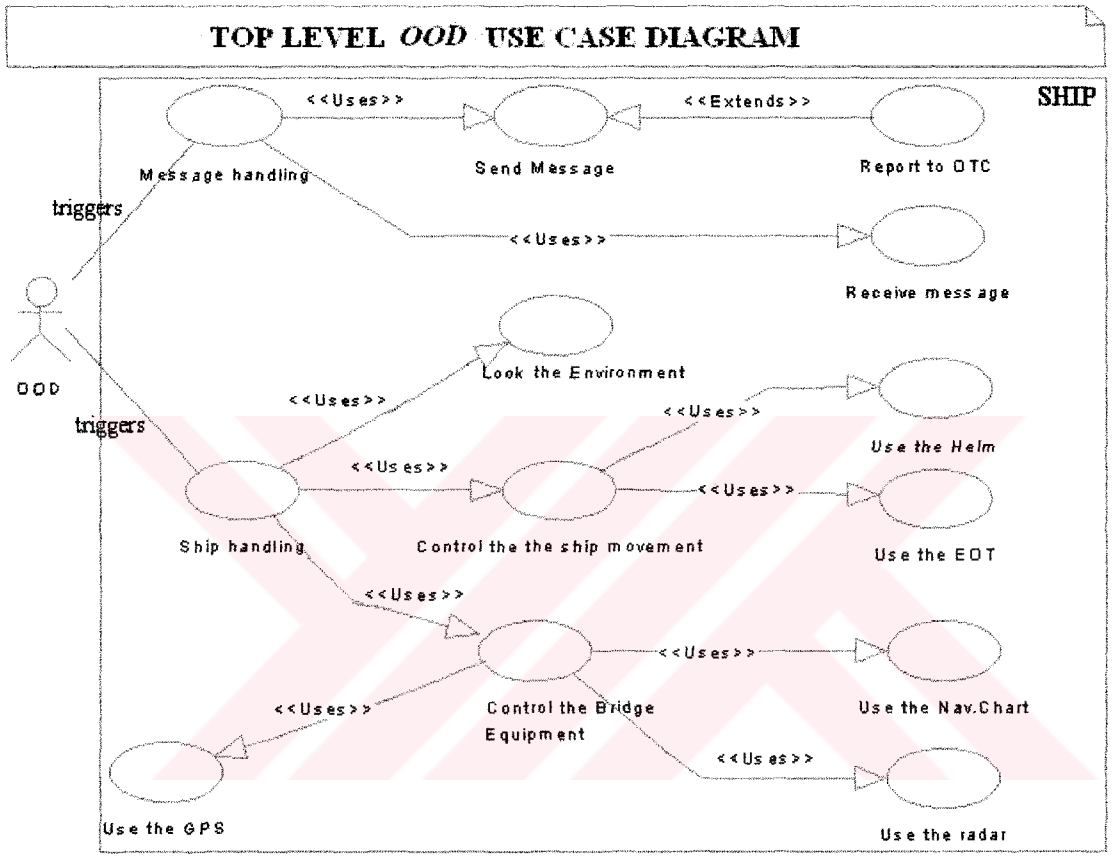


Figure B.2 Top Level OOD Use Case Diagram

B.2.3 TOP LEVEL OTC USE CASE DIAGRAM

The following diagram (Figure B.3) represents the use cases for the OTC. Here, the main use cases are also detailed.

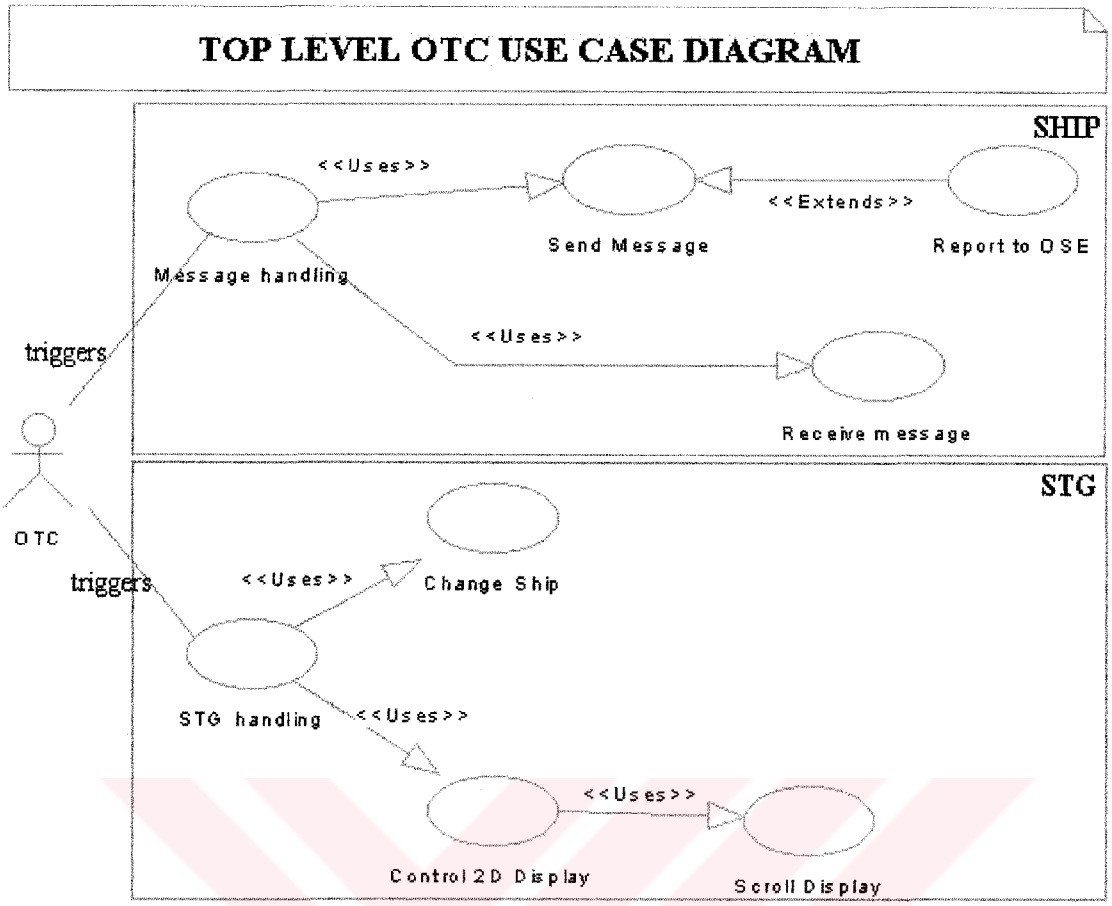


Figure B.3 Top Level OTC Use Case Diagram

B.3 INTERACTION DIAGRAMS:

Interaction diagrams describe how use cases are realized as interactions among societies of objects. Interaction diagrams come in two forms based on the same underlying information but each emphasizing a particular aspect of it:

- Sequence diagrams, and
- Collaboration diagrams

Sequence diagrams and collaboration diagrams express similar information but show it in different ways. *Sequence diagrams* show the explicit sequence

of messages and are better for real-time specifications and for complex scenarios. *Collaboration diagrams* show the relationships among objects and are better for understanding all of the effects on a given object.

SEQUENCE DIAGRAMS:

Sequence diagrams show interactions arranged in time sequence. The representation focuses on expressing interactions. A sequence diagram traces the execution of an interaction in time.

Here, the use of sequence diagrams corresponds to the documentation of use cases. It focuses on the description of the interaction, often in terms that are close to the user, without getting into the details of synchronization. In this case, the information carried by the arrows corresponds to events that occur within the application domain. At this stage of modeling, the arrows do not yet correspond to 'message broadcasts' in the sense of programming languages, and the difference between control flow and data flow is not generally established.

An *object* is represented by a rectangle and a vertical bar called the *object's lifeline*, which represents the existence of the object at a particular time. Objects communicate by exchanging messages, represented by horizontal arrows drawn from the message sender to the message recipient.

COLLABORATION DIAGRAMS:

Collaboration diagrams illustrate interactions between objects, using a static spatial structure that facilitates the illustration of the collaboration of a group of objects. Collaboration diagrams express both the context of a group of objects (through objects and links) and the interaction between these objects (by representing messages). Unlike a sequence diagram, a collaboration diagram shows the relationships among the object roles. On the other hand, a collaboration diagram does not show time as a separate dimension, so the sequence of messages and the concurrent threads must be determined using *sequence numbers*. *Collaboration* is a set of participants and relationships that are meaningful for a given set of purposes. Collaboration may be attached to an operation or a use case to describe the context in

which their behavior occurs. Collaboration may also be attached to a class's static structure. The implementation of collaborations brings in additional objects, which, although they do not belong to the application domain, are necessary for its operation. These objects generally implement the interface between the system and the user. Collaboration may be expressed at different levels of granularity. A coarse-grained collaboration may be refined to produce another collaboration that has a finer granularity.

B.3.1 "OOD SHIP HANDLING" INTERACTIONS

Table B.2 summarizes all interactions between OOD and the ship within 'ship handling' use case.

Table B.2 OOD Ship Handling Interactions

USE CASE	MAIN SCENARIOS/INTERACTIONS
Using the helm	Turning the ship to the R/L by ordering a specific rudder angle
	Turning the ship to the R/L by ordering a new ship's heading (NOT IMPLEMENTED)
Using the EOT	Moving the ship forward
	Stopping the ship
	Moving the ship backward
Using the GPS	Looking the GPS display
Using the radar	Looking the radar scope
	Changing radar scale
	Measuring the distance of a RADAR echo
	Measuring the bearing of a RADAR echo
Using the nautical chart	Selecting the chart (NOT IMPLEMENTED)
	Drawing routes (NOT IMPLEMENTED)
Looking the environment	Looking ahead sector
	Looking R/L sector
	Looking astern sector

USE CASE: USING THE HELM

Turning the ship to the right/left (R/L) by ordering a specific rudder angle:

The figure B.4 shows the interaction between the OOD and helm to turn the ship by ordering a specific rudder angle. *The `KNOX` object* is a composite object, and *the `helm` object* is one of the components of *the `KNOX` object* but it is shown for the purpose of depicting the interactions more clearly. The helm object is composed of three objects `Helm_UI`, `Helm`, and the `Helmsman`. The properties of these objects will be described in design section. The helmsman object is designed to be used in "turning the ship to the right/left by ordering a new ship's heading" interaction. The blue part depicts the user interface interactions in terms that are close to the user. This part encapsulates the interactions between user and user interface such as using the RID slider widget and keyboard. RID stands for rudder indicator degree.

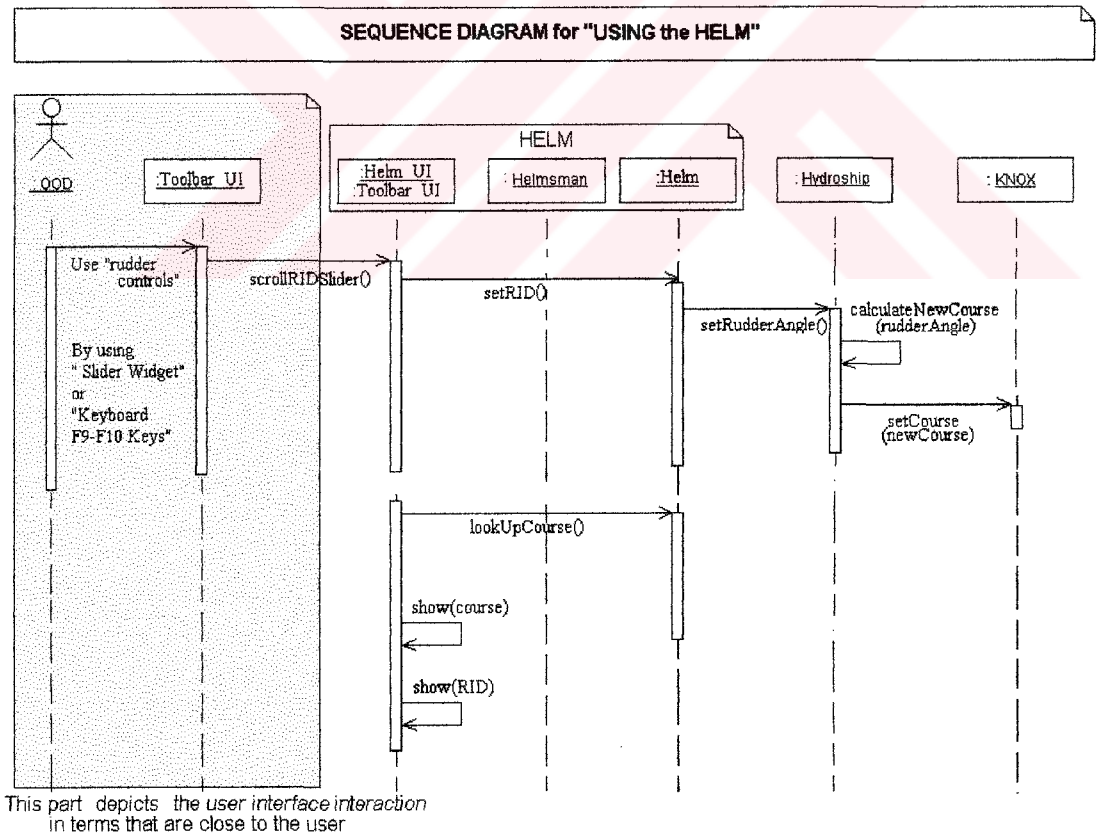


Figure B.4 the Sequence Diagram for "Using the helm"

The figure B.5 shows the function calls between objects after the user OOD use the rudder to turn the ship. The OOD triggers a set of functions resulting the ship starts to turn until OOD orders another helmsman command. While simulation goes on, the bridge displays continuously update their values to reflect the correct information to the user. In order the *Helm_UI* object to display the RID and course values, it should read the values from the *Helm* object, which models the behavioral characteristics of the helm. This situation is depicted in the Figure B.6. This set of interactions continuously goes on during simulation without the need of a triggering event.

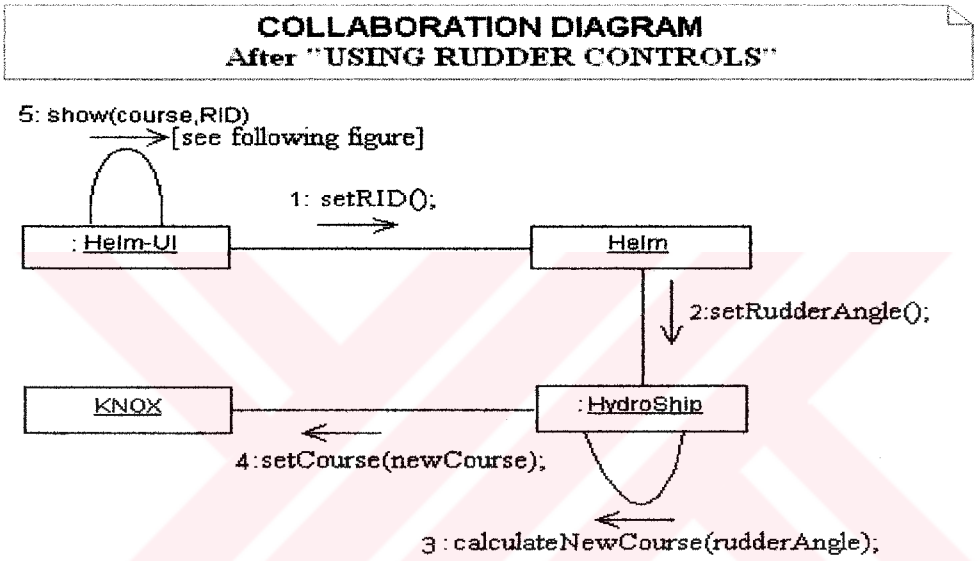


Figure B.5 Collaboration Diagram after "Using Rudder Controls"

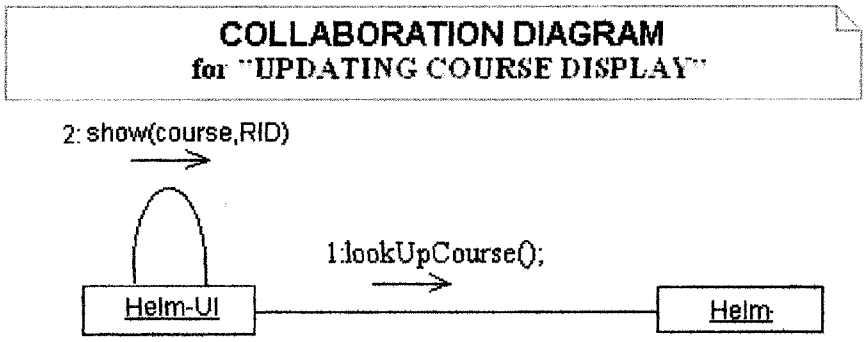


Figure B.6 Collaboration Diagram for "Updating Course Display"

Turning the ship to the right/left by ordering a new ship's heading:

Not Implemented.

USE CASE: USING THE EOT

Such as using the helm, there are some scenarios in using the EOT to control the ship's movement.

Moving the ship forward:

- OOD gives one of EOT commands: *"Ahead 1/3-2/3-standart-full-flank"*,
- EOT transmits in RPM (revolution per minute) the shaft turns to the engine control room of the ship,
- Ship increases or decreases its speed.

Stopping the Ship:

- OOD issues EOT command "Stop" by using the EOT Control Buttons,
- EOT transmits this command to ship,
- Ship stops.

Moving the Ship Backward:

- OOD gives one of the EOT commands: "*Back 1/3-2/3-full*",
- EOT transmits shaft turns in RPM to the engine control room,
- Ship goes back accelerating or de-accelerating.

The diagram in the figure B.7 is suitable for the whole three scenarios. The OOD uses the EOT radio button or keyboard to trigger a set of interactions resulting movement of the ship.

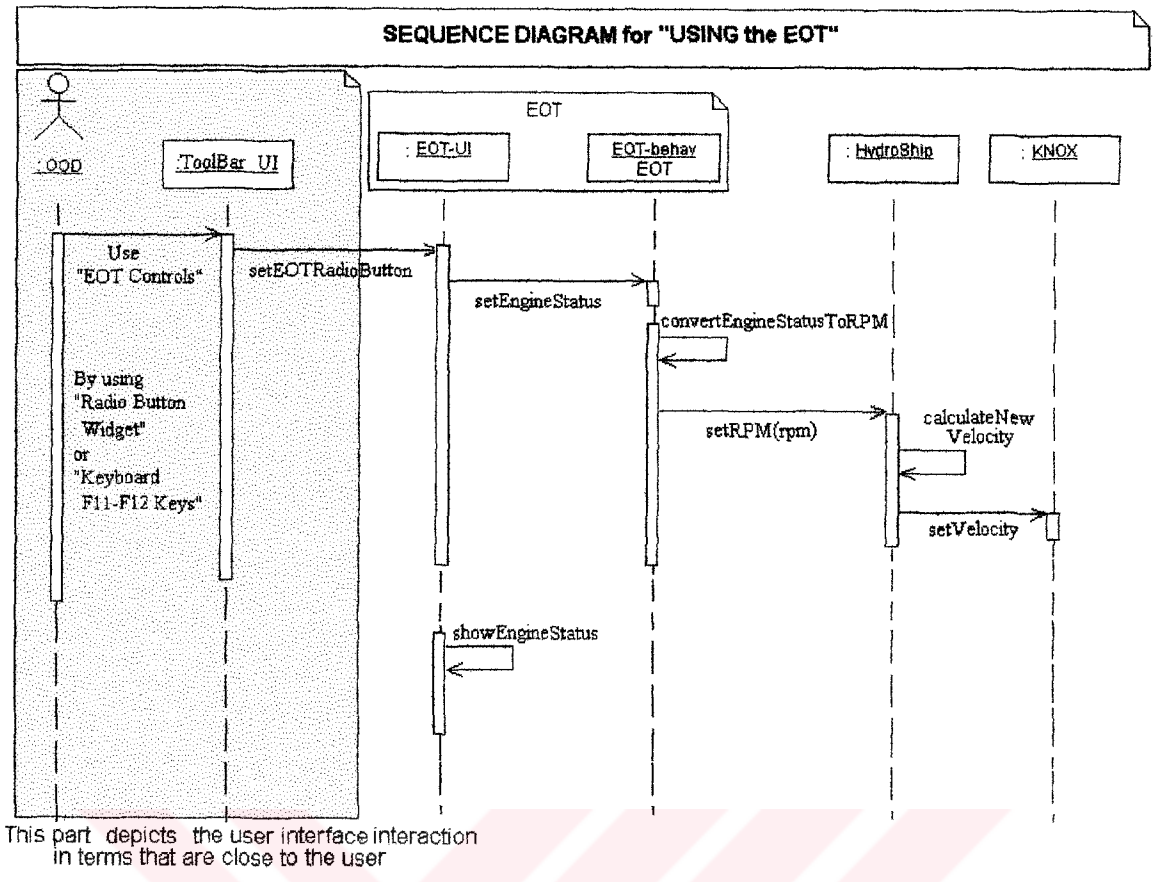


Figure B.7 Sequence Diagram for "Using the EOT"

The collaboration diagram in the figure B.8 depicts the interactions by emphasizing objects after the OOD triggered the use-case.

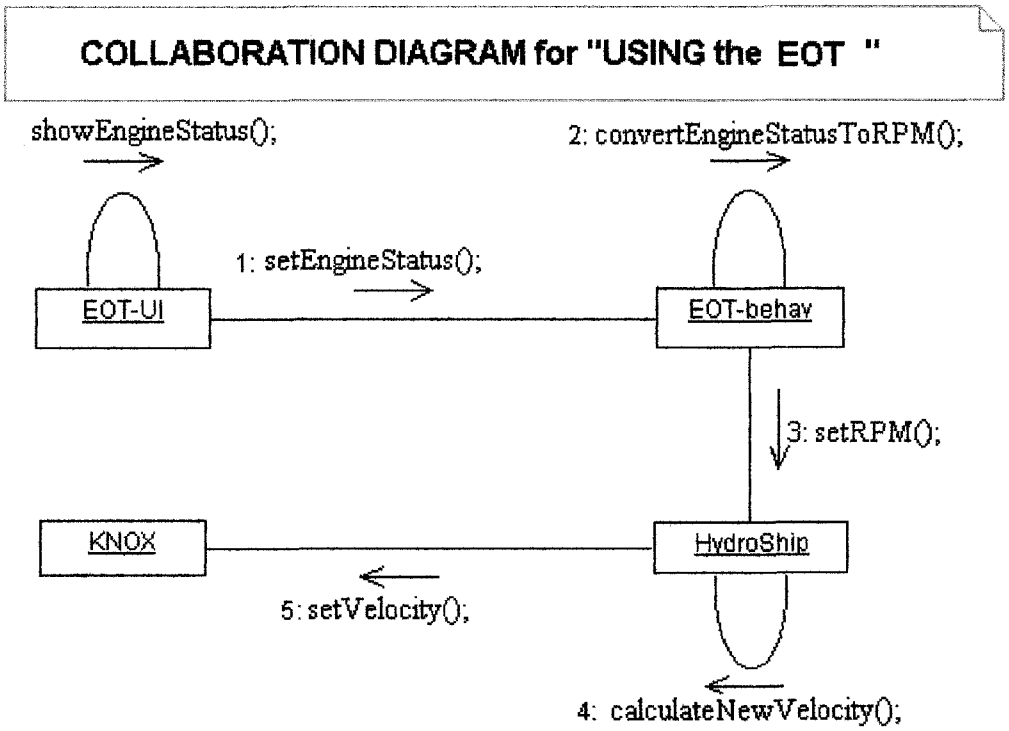


Figure B.8 Collaboration Diagram for "Using the EOT"

USE CASE: USING THE GPS

Looking the GPS display:

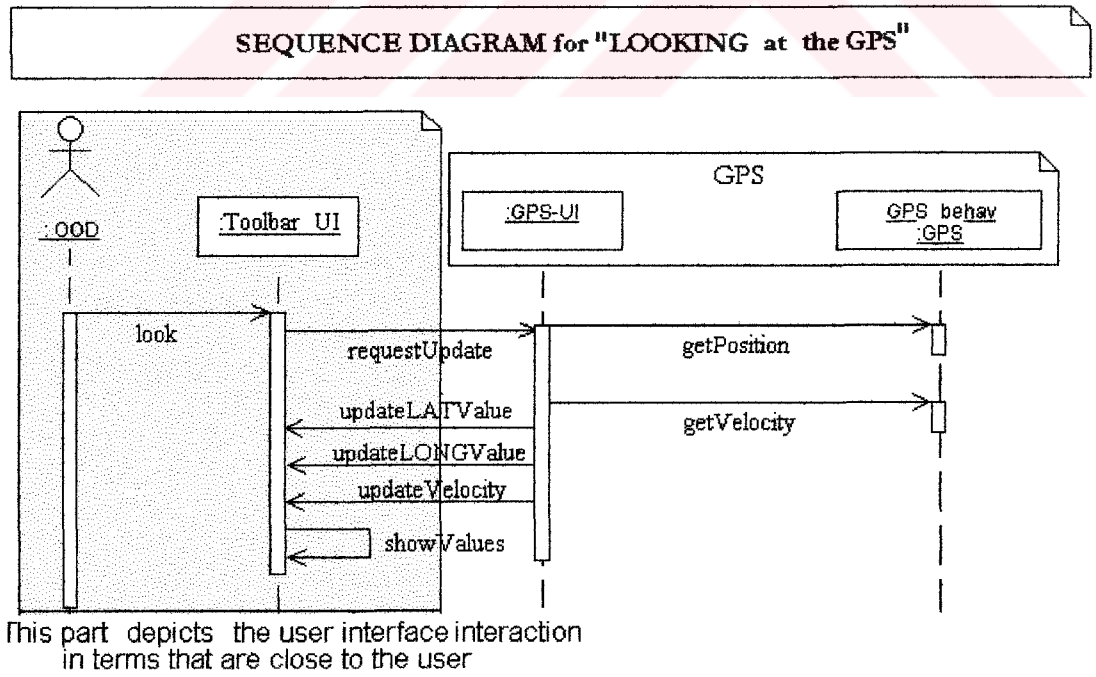


Figure B.9 Sequence Diagram for "Looking at the GPS Display"

GPS widget update is being done continuously during simulation to reflect the current values. GPS widget displays Latitude, longitude, and velocity of the ship. The GPS_UI object gets these values from the GPS object, which models behavior of the GPS equipment. GPS object formats the position in a user-readable way (e.g. 120° 24' 45" E).

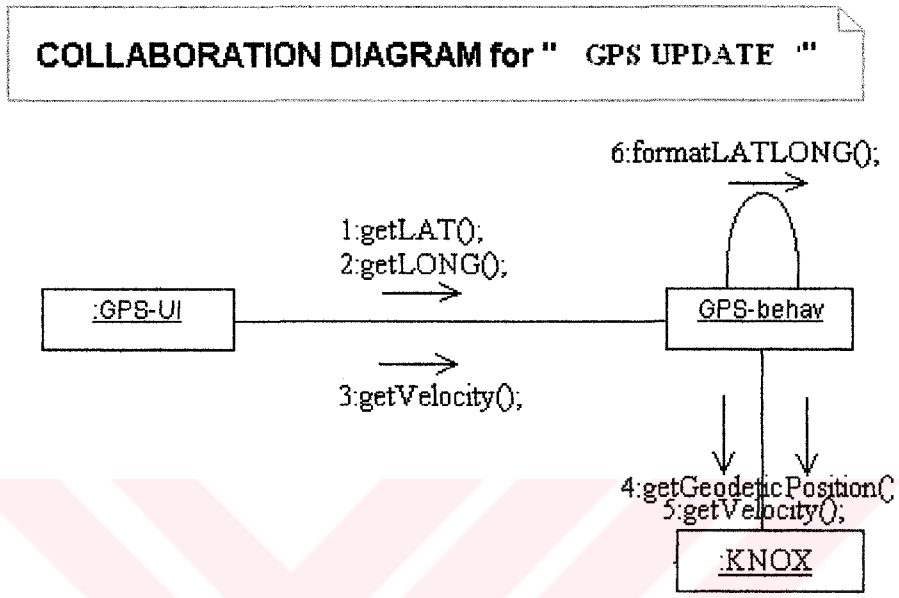


Figure B.10 Collaboration Diagram for "Updating GPS values"

USE CASE: USING THE RADAR

Radar display is always on screen. There are three use-cases related to radar use:

- The OOD can look at Radar display to see the geographic area and other ships around and the values of distance and bearing of the ships,
- The OOD can change radar scale to increase or decrease the coverage area,
- The OOD can measure the distance and bearing of other ships around.

Looking at the RADAR Scope:

The OOD can easily look at geographic area and displays without distracting his/her view of the surrounding scene.

Changing the RADAR Scale:

The OOD can change scale using "scale control buttons" or "keyboard +/- keys". Radar object is responsible to calculate the coverage area. The figure B.11 presents the sequence diagram.

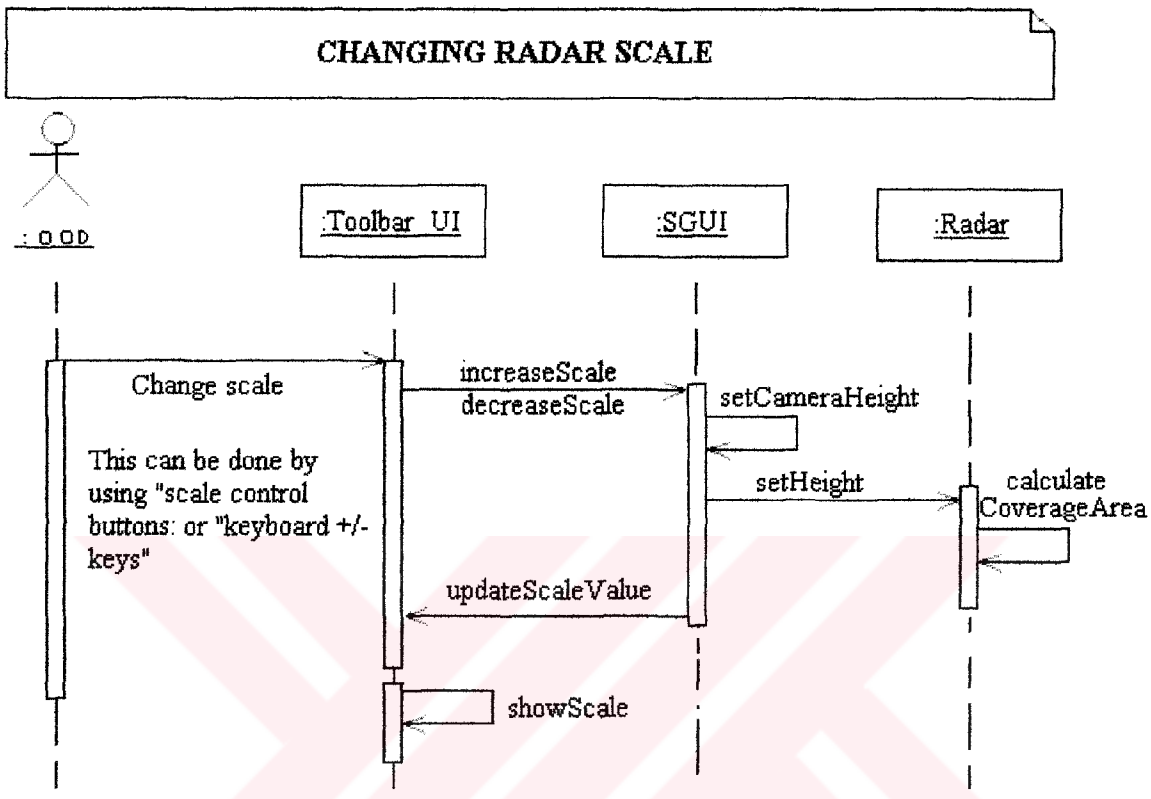


Figure B.11 Sequence Diagram for "Changing the Radar Scale"

Measuring the Distance of a RADAR Echo:

Figure B.12 shows the interactions to measure the distance of a radar echo. These interactions are:

- The OOD selects a ship, which is displayed on radar screen,
- Radar object calculates the distance from local ship to the selected ship,
- "Distance textbar widget" on Tool Bar shows the distance value.

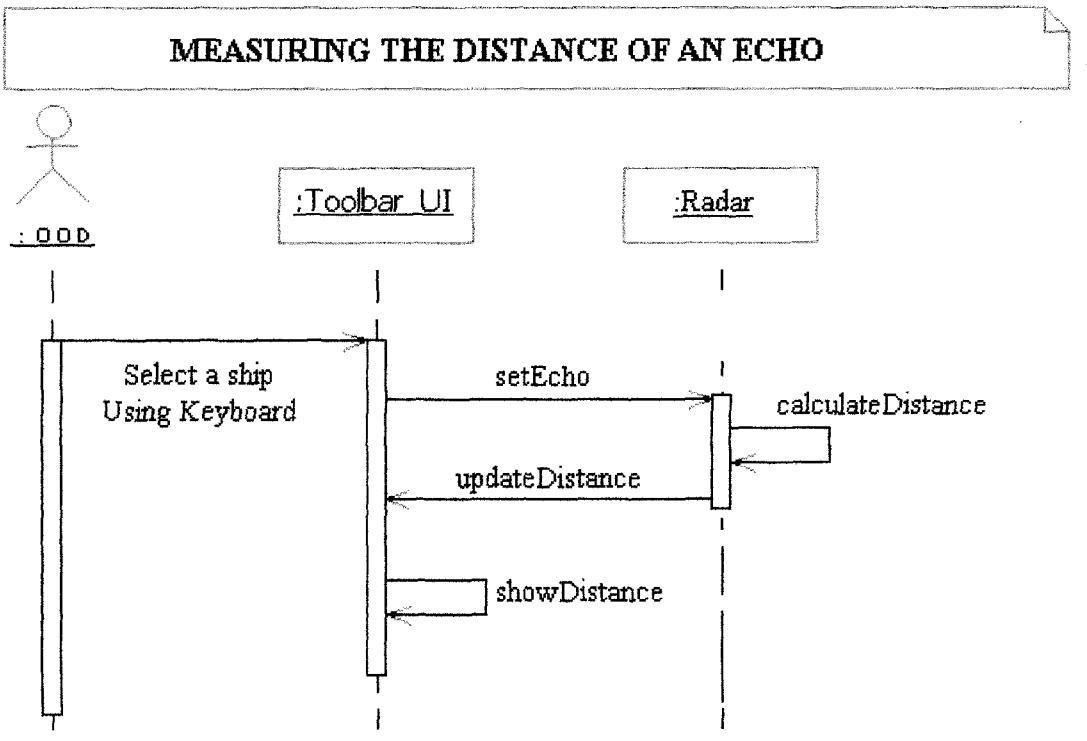


Figure B.12 Sequence Diagram for "Measuring the Distance of a Radar Echo"

Measuring the Bearing of a RADAR Echo:

- The OOD selects a ship, which is displayed on radar screen,
- Radar object calculates true bearing from local ship to the selected ship,
- "Distance textbar widget" on Tool Bar shows the bearing value.

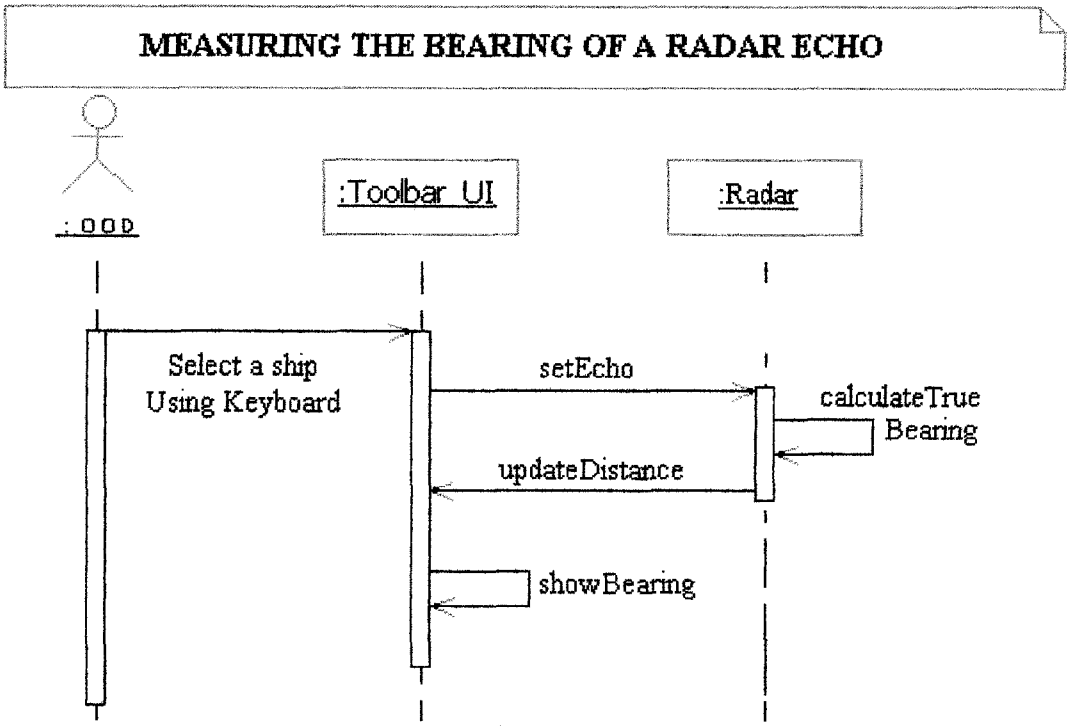


Figure B.13 Sequence Diagram for "Measuring the Bearing of a Radar Echo"

USE CASE: USING THE NAUTICAL CHART

Selecting the chart:

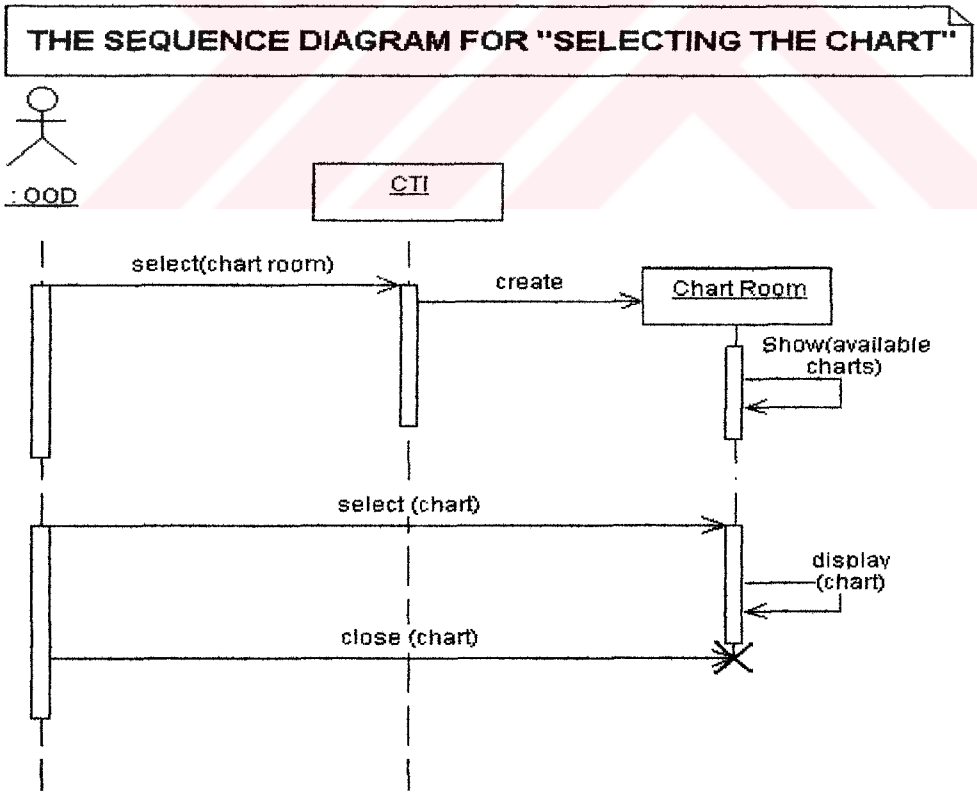


Figure B.14 Sequence Diagram for "Selecting the Chart"

Drawing routes:

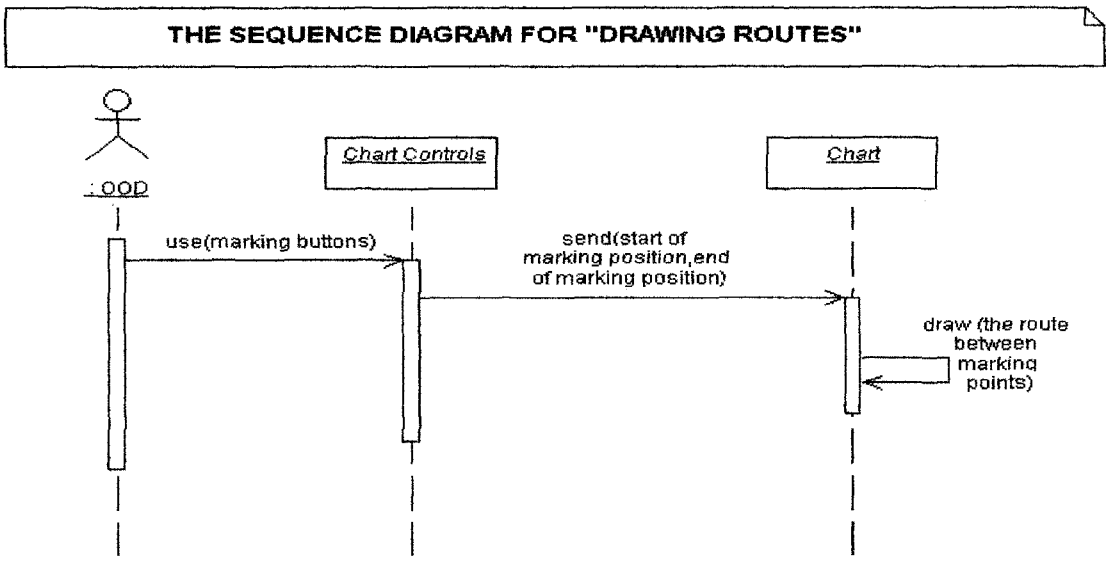


Figure B.15 Sequence diagram for "Drawing Routes"

USE CASE: LOOKING AT THE ENVIRONMENT

This use case provides the capabilities for OOD to see the outer environment by looking the head sector of the ship, by looking at the right/left sector of the ship or by looking the back sector of the ship from the center point of the bridge.

Looking ahead, R/L(right/left), or astern sector:

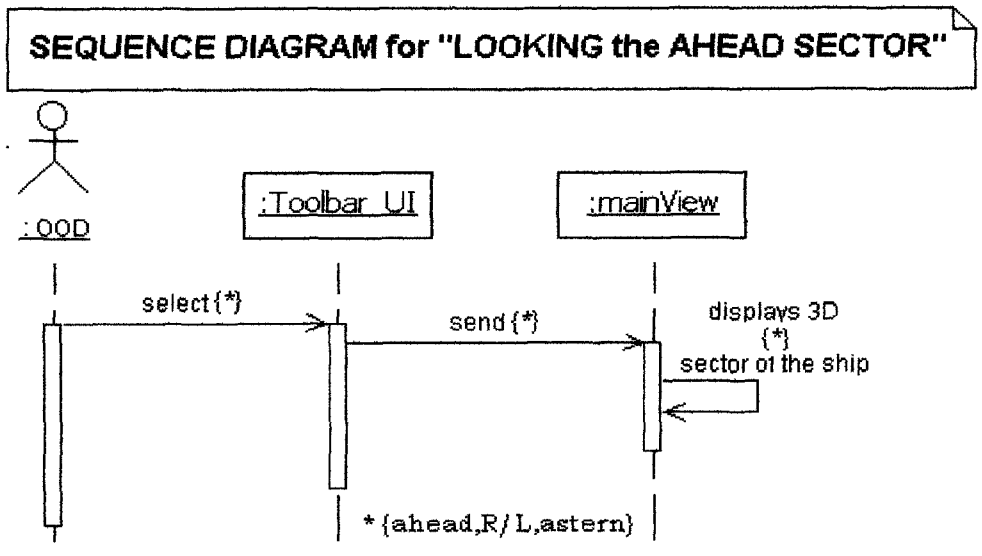


Figure B.16 Sequence diagram for "Looking Ahead, R/L, or Astern Sector"

B.3.2 "OOD MESSAGE HANDLING" INTERACTIONS

Table B.3 summarizes all interactions between OOD and the ship within message handling use case and it also shows the sub-use cases.

Table B.3 OOD Message Handling Interactions

USE CASE	MAIN SCENARIOS/INTERACTIONS
Sending Message	Sending the message
Receiving Message	Seeing the incoming message

USE CASE: SENDING MESSAGES

Sending the message:

Before the OOD starts to type message, he/she should select the SEND_MODE. After completing the message, the OOD presses "enter" key to send the message. System

- Checks the header,
- Adds it to send-queue,
- Adds subject,
- Sends the message as an interaction-class.

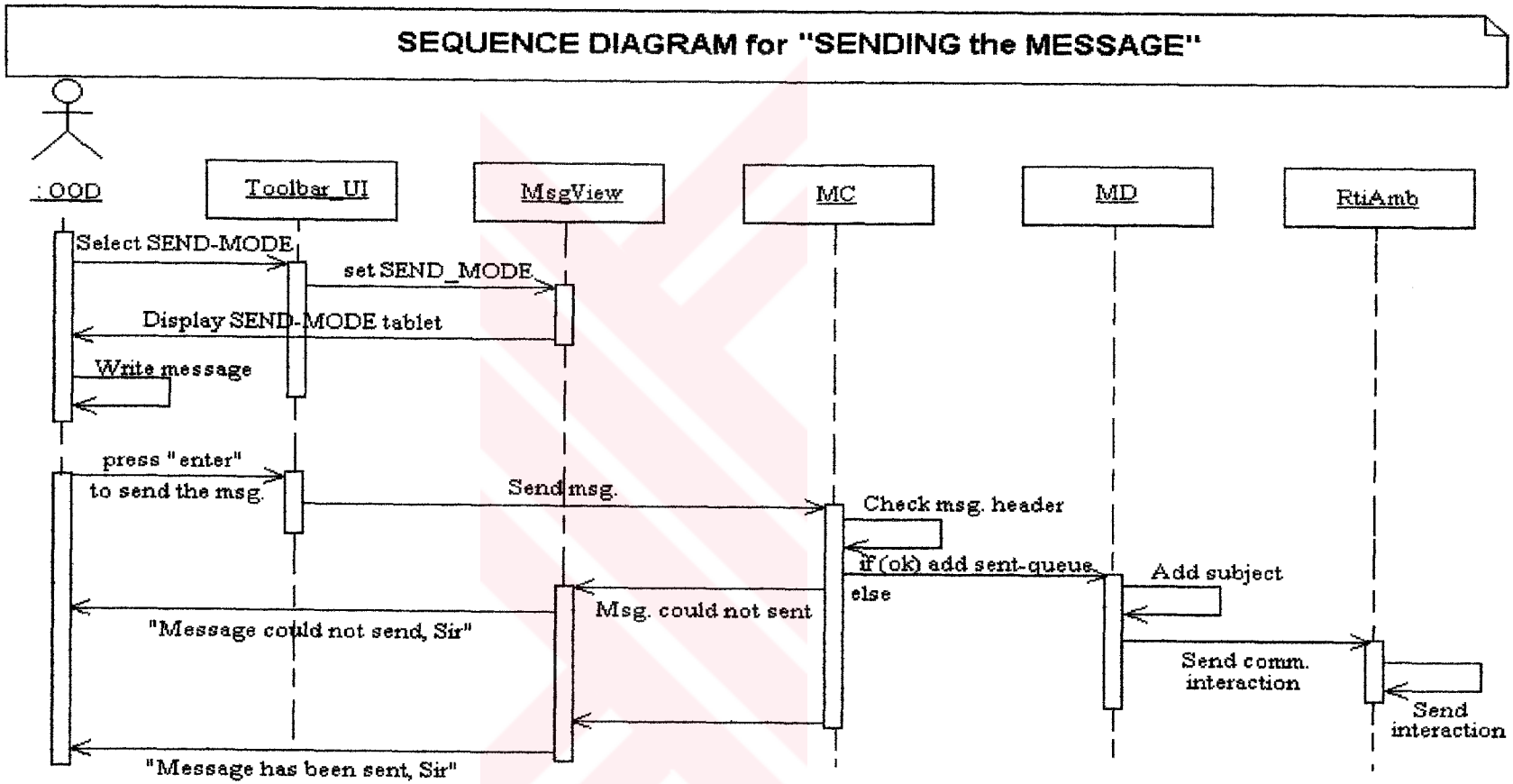


Figure B.19 Sequence diagram for "Sending the Message"

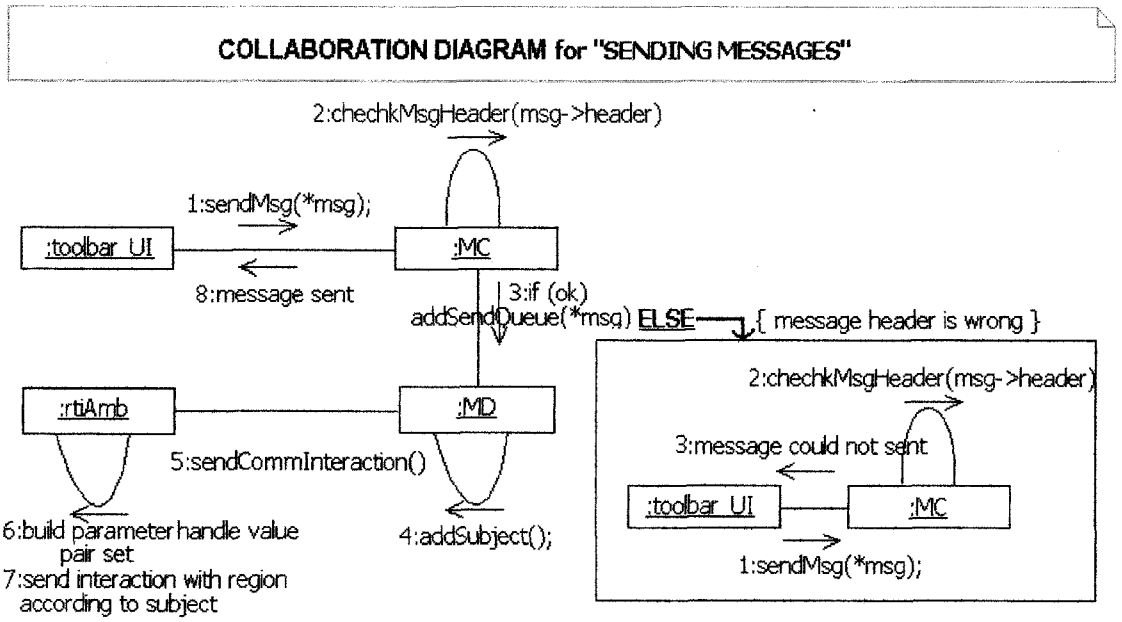


Figure B.20 Collaboration Diagram for "Sending Messages"

USE CASE: RECEIVING MESSAGES

Seeing the incoming message

After "FedAmb receive interaction callback" is triggered, system decodes the incoming interaction, filters the message and indicates that a new message has arrived.

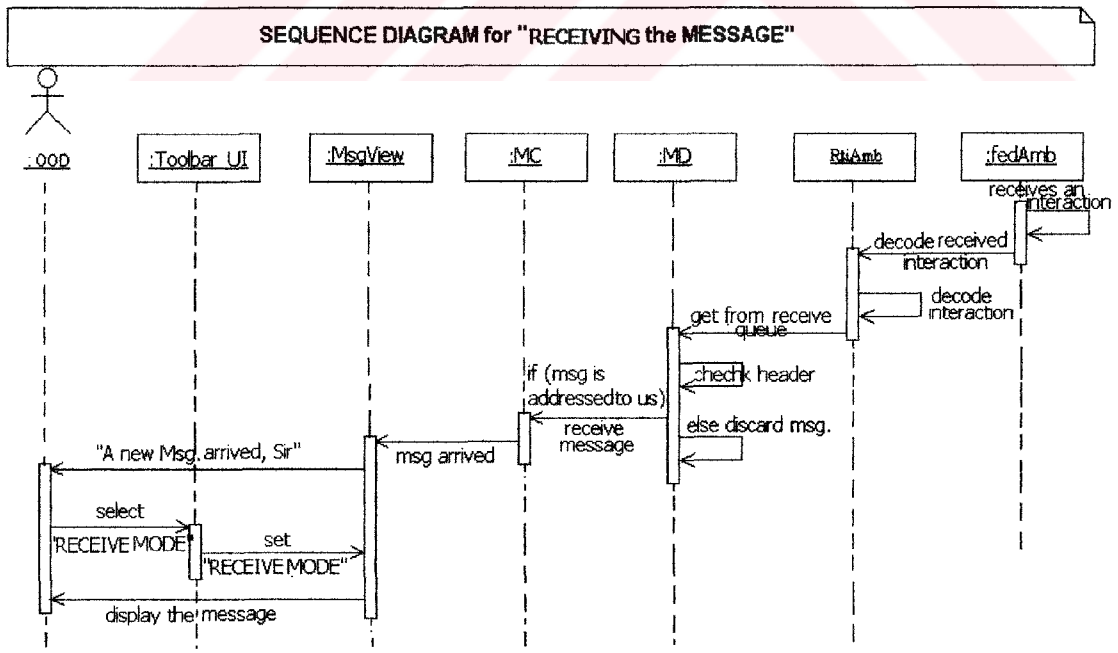


Figure B.21 Sequence diagram for "Seeing the Incoming Message"

B.3.3 "OTC STG HANDLING" INTERACTIONS

Table B.4, summarizes all scenarios between OTC and STG.

Table B.4 OTC STG Handling Interactions

USE CASE	MAIN SCENARIOS/INTERACTIONS
Changing Ship	Initially selecting the ship (NOT IMPLEMENTED)
	Changing the ship (NOT IMPLEMENTED)
Controlling 2D View	Scrolling Display
	Zooming in
	Zooming out

USE CASE: CHANGING THE SHIP

Changing the Ship:

The OTC can change the ship in run-time, though this use-case did not implemented.

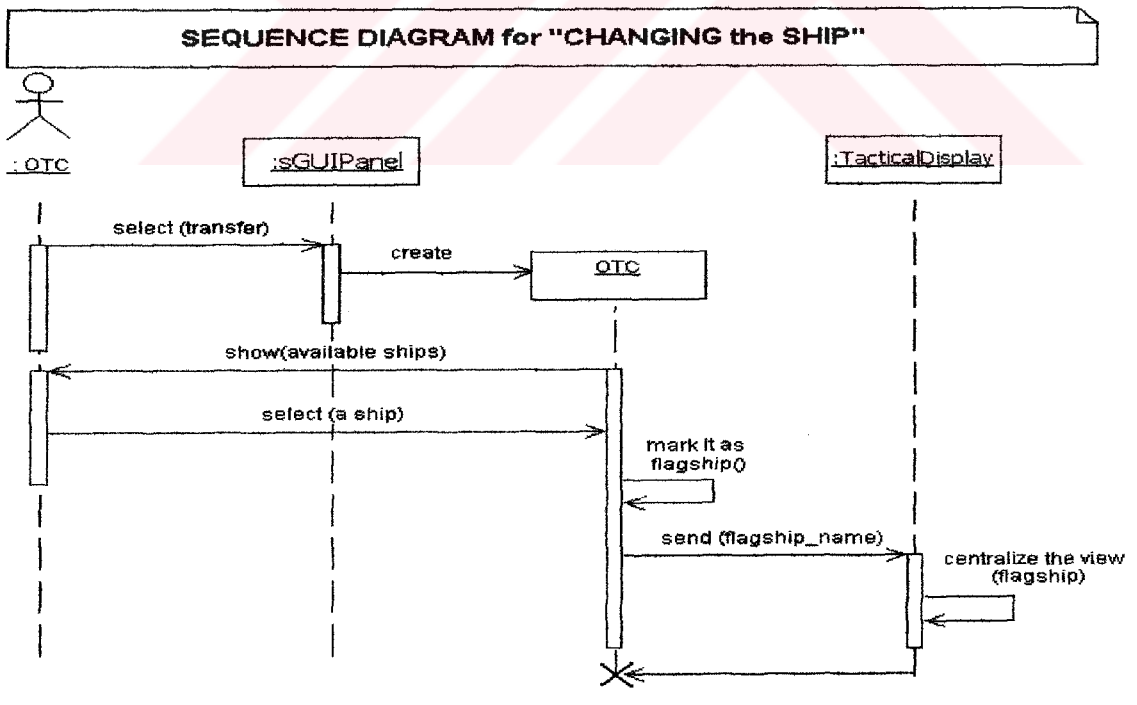


Figure B.22 Sequence Diagram for " Changing the Ship"

USE CASE: CONTROLLING 2D DISPLAY

Scrolling Display:

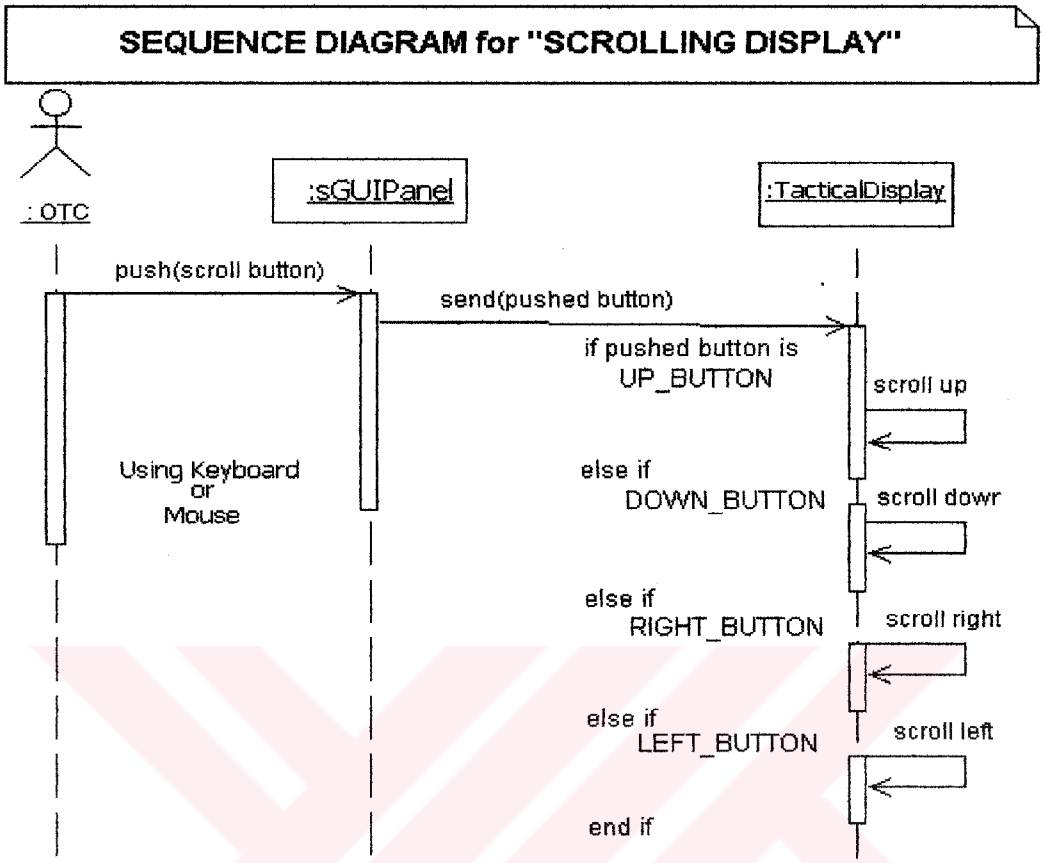


Figure B.23 Sequence Diagram for "Scrolling the Display"

Zooming In:

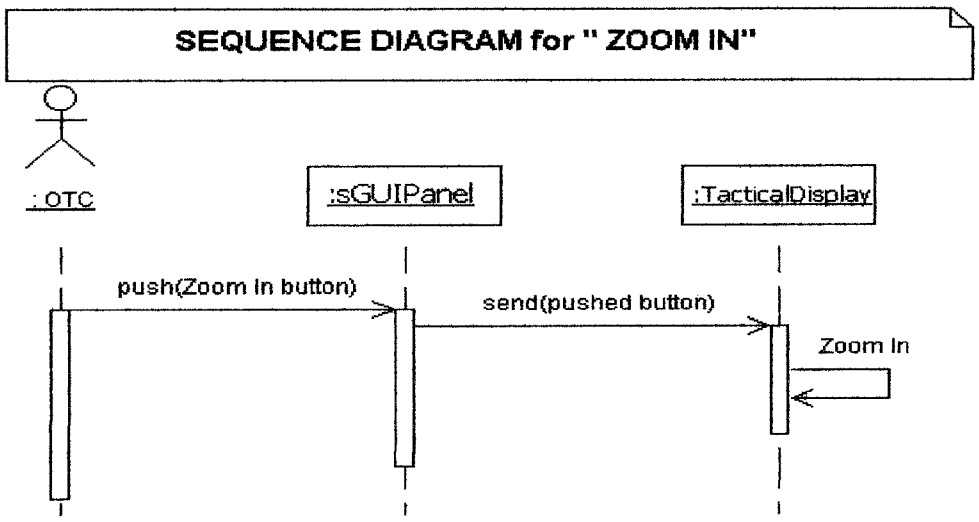


Figure B.24 Sequence Diagram for "Zoom In"

Zooming Out:

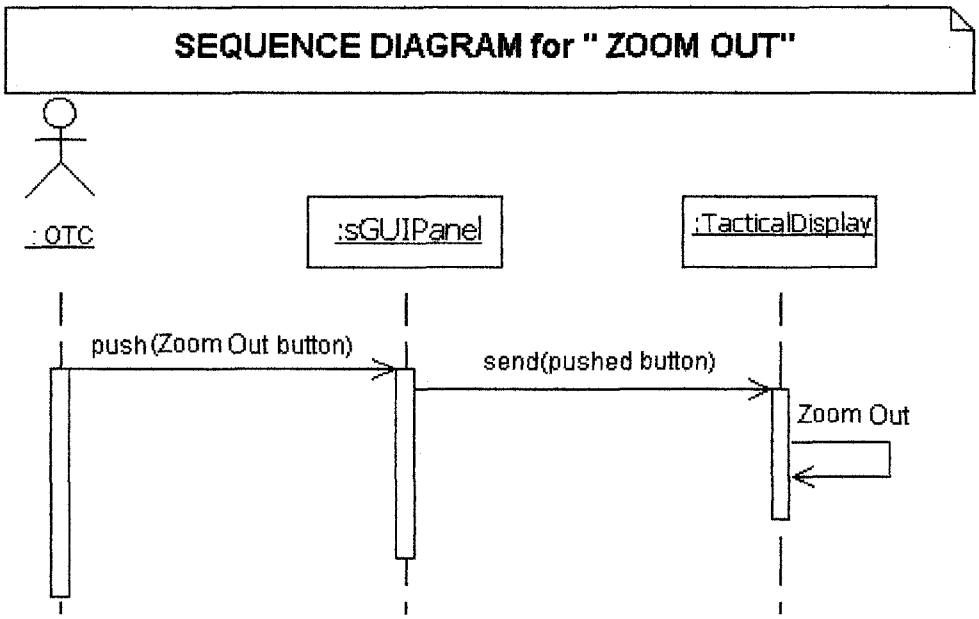


Figure B.25 Sequence Diagram for "Zoom Out"

B.3.4 "OTC MESSAGE HANDLING" INTERACTIONS

All OTC message handling interaction diagrams is the same with OOD message handling interaction diagrams, if we assume that these changes are being done:

- Toolbar_UI class is being replaced by sGUIPanel class,
- OTC plays OOD role for sending and receiving messages.

Note that OTC uses the message center of the ship, which he is onboard.

APPENDIX – C

OBJECT ORIENTED DESIGN

C.1 CLASS DIAGRAMS

A *Class Diagram* shows the static model elements, such as classes, interfaces, and their relationships in the logical view of the system.

C.1.1 KNOX FEDERATE (KNOXFd) CLASS DIAGRAMS

At this section, the internal structure of KNOXFd is being detailed and designed. The overall class diagram for Knox federate is being depicted in the figure C.1.

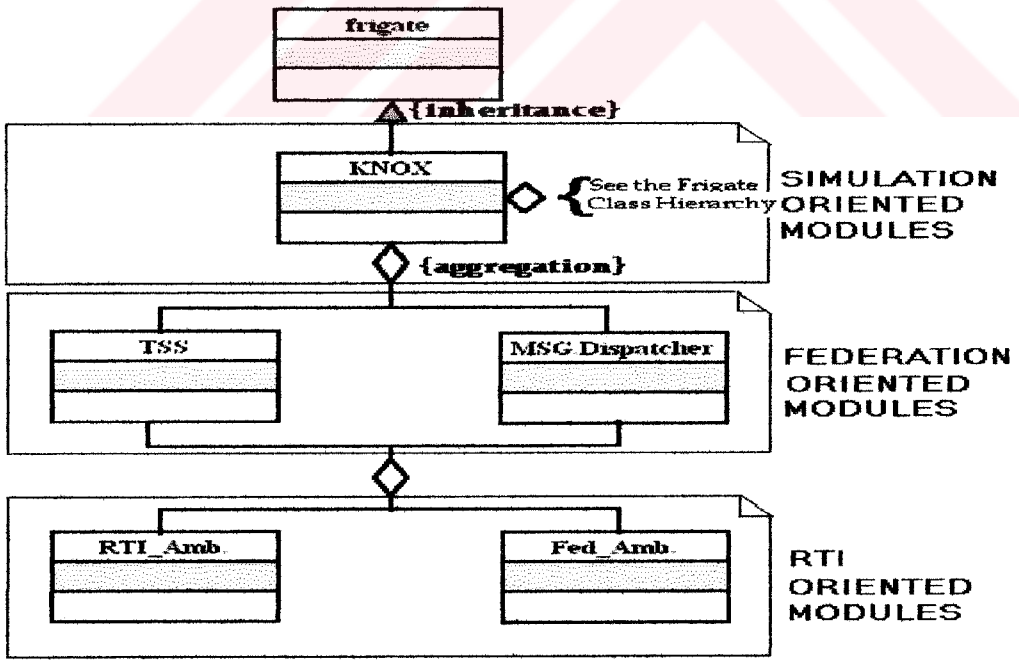


Figure C.1 KNOX Class Diagram

Although, the federation and RTI oriented class hierarchies are clear, it is worth to look into details of simulation oriented class hierarchy. There exist two main class hierarchies in the simulation/federate oriented modules layer: the *Frigate Class Hierarchy and Single Graphical User Interface (sGUI) Class Hierarchy*.

A *well-defined interface* (this is class interface, not user interface) is formed to achieve the communication between the two classes.

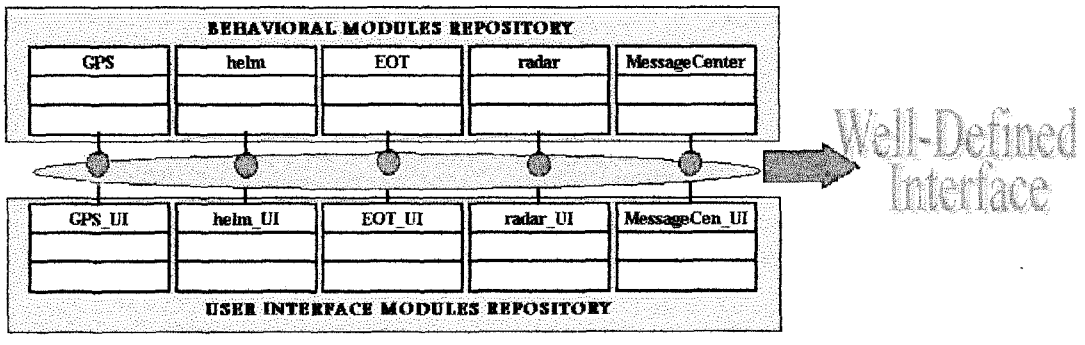


Figure C.2 Well-Defined Interface Between Modules

A class interface is being represented using small circles connected with a line to the class in the UML notation.

The abstract superclass is central to the design of a derivation hierarchy. Regardless of how deeply nested and complex the hierarchy becomes, the relationship between classes is maintained by the common set of class members inherited from the abstract superclass. A class that serves as the root class of a derivation hierarchy is sometimes referred as an *abstract superclass* [S97]. `Frigate` *and* `viewWindow` classes play the role of an abstract superclass. For example, the `frigate` class serves as the root base class of all the frigate ship types; such as KNOX class frigate or MEKO 200T class frigate, etc., although only KNOX class frigate is implemented.

Let's briefly review some important classification tools in UML. *Inheritance* is a technique offered by object oriented programming languages to construct a class from one or more other classes, sharing attributes, operations and

sometimes constraints within hierarchy of classes. *Aggregation* is a relationship where relationship is between a whole and its parts. *Multiplicity* defines how many objects participate in a relationship.

FRIGATE CLASS HIERARCHY

The frigate class hierarchy is depicted in the figure C.3. The frigate class hierarchy has nothing to do with user; it focuses to the functionality of a virtual ship and its surroundings.

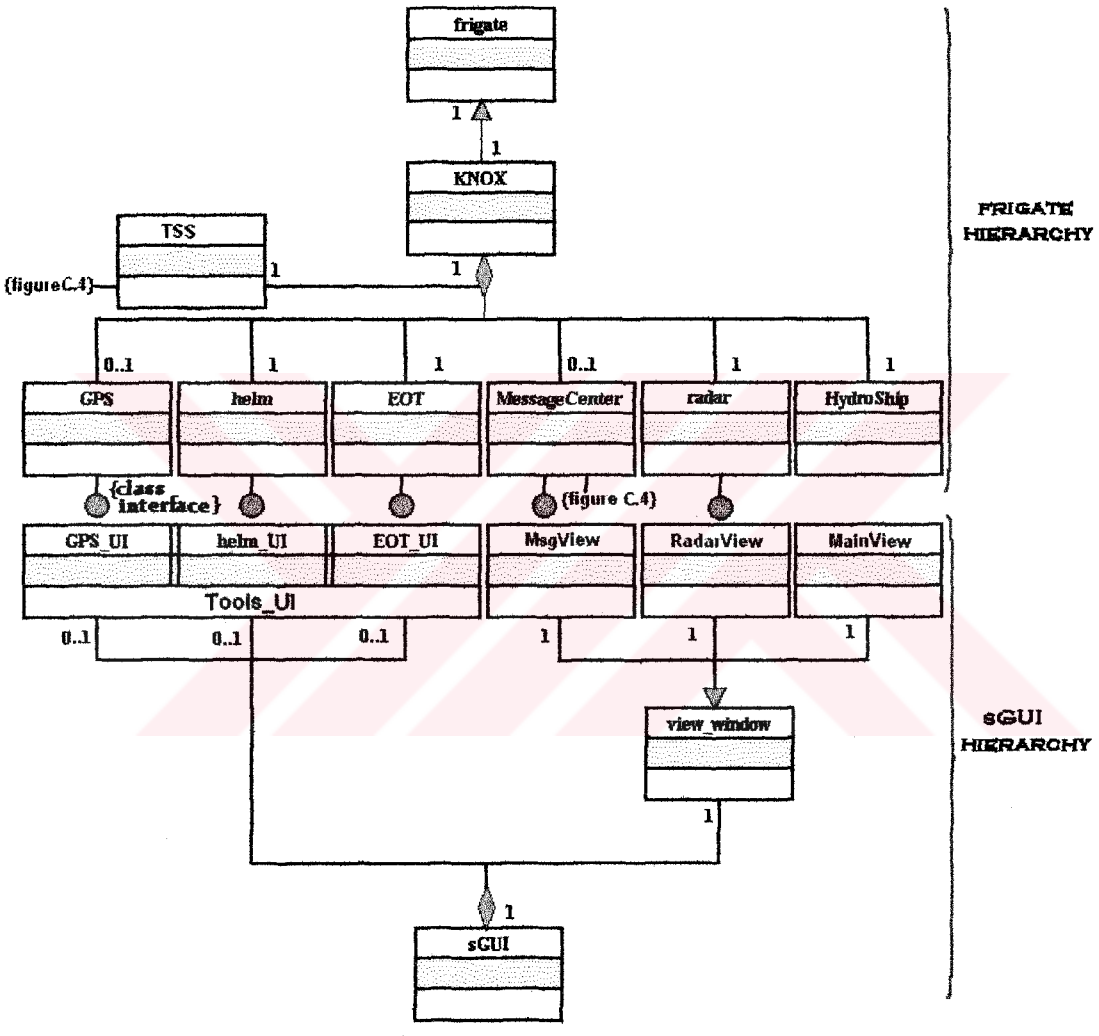


Figure C.3 Frigate and sGUI Class Hierarchies

The root of the Hierarchy is an abstract class, which is called `frigate` class. `KNOX` class is inherited from `frigate` class. `KNOX` class is composed of behavioral models of bridge equipment. This relationship is being classified as aggregation (represented by a diamond in the figure C.3).

Figure C.4 completes the description of the frigate class structure. *TSS* class and *Message Center* class connect the frigate class hierarchy into federation-oriented modules.

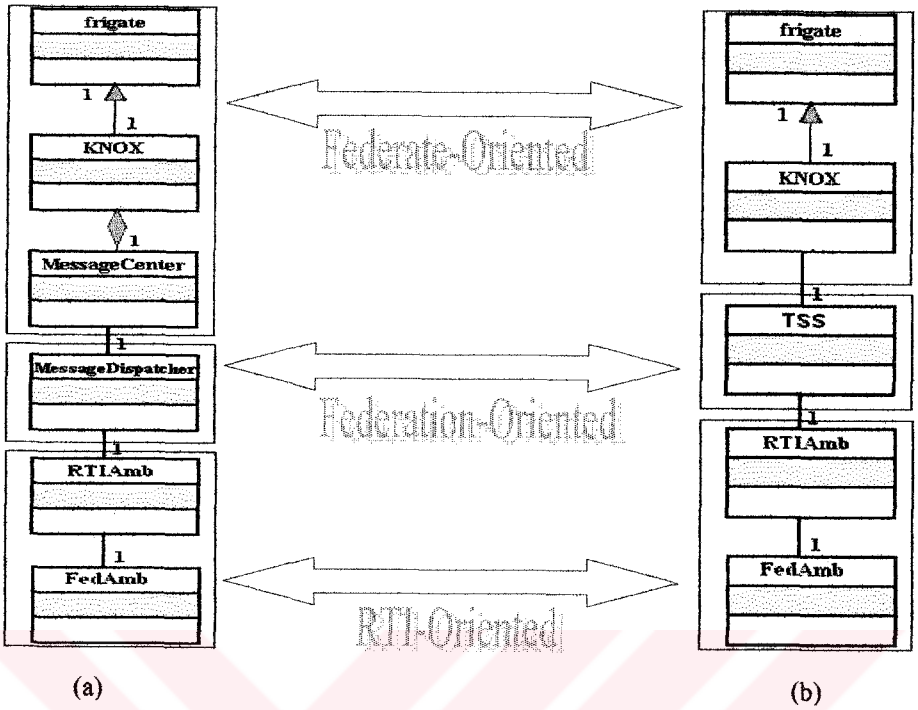


Figure C.4 Frigate Class Hierarchy

KNOX class consists of:

- `Helm, GPS, EOT, radar, message center` classes to provide all the behavioral functionality of the system,

- *HydroShip* class, which implements simple hydrodynamics for the ship movement,

- *TSS* class, which implements the dead-reckoning algorithms, and provides the dead-reckoning data (position, speed, course), and tracks accurately the other virtual environment entities participating in the federation (e.g. other ships),

- *RTIAmbassador* class to manage the RTI-related work (e.g. federation management, declaration management),

- *FEDAmbassador* class to get callbacks.

sGUI CLASS HIERARCHY

sGUI class hierarchy deals with the graphical user interface part of the system (e.g. the GUI of helm). It directly interacts with the user. It is a bridge between the user and the system. *sGUI* class hierarchy consists of:

- Provides all the interface functionality of the system by utilizing the user interface modules (e.g. *helm_UI, radar_UI*),

- Provides high level window management functions, which are independent of operating system window management functions, by utilizing Performer Utility classes,

- Renders 3D virtual environment.

RadarView draws the radar display. *MsgView* draws the message center and controls the keyboard to handle message writing. *MainView* is responsible from the main 3D views of the OOD (e.g. ahead view, right/left view).

C.1.2 OTC FEDERATE (OTCFd) CLASS DIAGRAMS

The structure and logic of OTC federate class hierarchy is similar to KNOX federate. Again, I divide the hierarchy structure into two groups: the graphical and behavioral hierarchies.

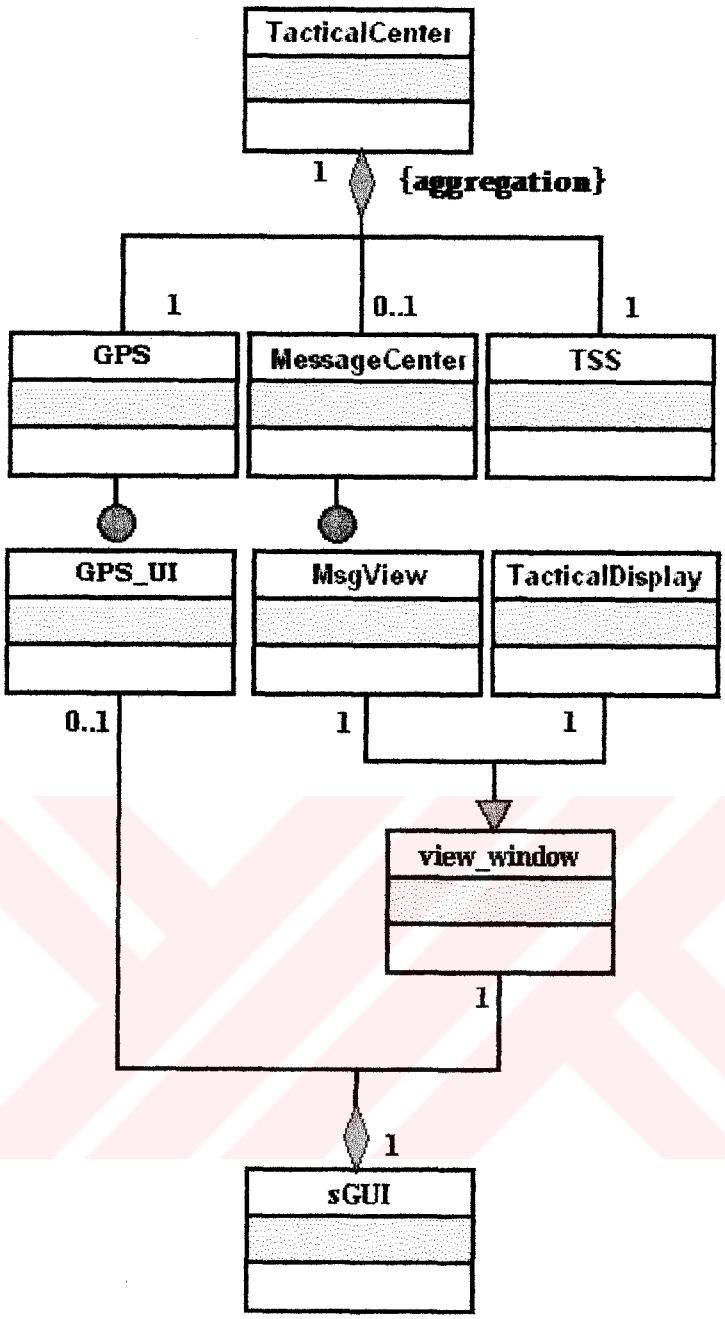


Figure C.5 OTC Federate Class Diagram

Figure C.5 depicts the OTCFd class diagram. *TacticalCenter* class consists of three sub-classes: *GPS*, *TSS*, and *TacticalDisplay*. The design aim of *GPS* and *TSS* classes are the same with KNOXFd classes. *TacticalDisplay* is designed for creating the bird's eye view, constructing the scene graph, and rendering the scene.

C.1.3 ENVIRONMENT FEDERATE (EnviFd) CLASS DIAGRAM

Environment Federate class structure is relatively simple. Figure C.6 presents the class diagram. *Envi* class is designed for handling command line interface interactions (e.g. creating fog, switching time of day). *RTIAmb* and *FedAmb* are responsible from RTI communication.

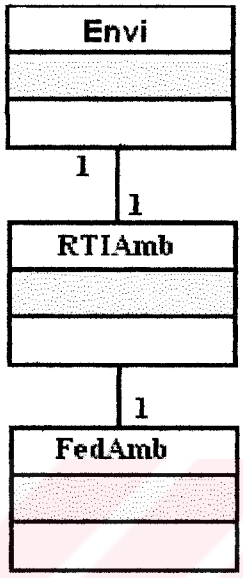


Figure C.6 Environment Federate Class Diagram

C.2 MAIN CLASSES

Classes are found by examining the use case diagrams. The member functions are found by examining the interactions in use cases and collaboration diagrams.

The first part gives the class name and the second part presents the class attributes, while third part presents the class functions. Remember that public member functions are constructing the class interface.

The signs preceding the attributes and functions are:
"+" means *public* so that the attribute or function is visible to other classes,
"-" means *private* so that the attribute or function is only visible to the class,

"#" means *protected* so that that the attribute or function is visible to the derived classes.

Only a representative selection of classes are shown here.

C.2.1 KNOX FEDERATE CLASSES

Frigate Class (abstract):

Frigate
geodetic: geodetic_pos_type // {LAT, LONG} - geocentric: geocentric_pos_type // {x, y, z} # course: course_type # velocity: velocity_type
+ returnCourse(): course_type + returnVelocity(): velocity_type +getPosition() :geocentric_pos_type

HydroShip Class:

Hydroship
- the_KNOX: KNOX * - rudder_angle: float - new_course: course_type - newVelocity: velocity_type - orderedRPM: int - presentRPM: int - RPMoffset: int - OldVTime,oldPTime: float // time dor pos./vel. Calculation - UnitDistance: float // distance taken in unit time
HydroShip() ~HydroShip() + set_the_KNOX(KNOX *): void + setRudderAngle(float): void + setRPM(int): void +calculateNewVelocity(): void

+ calculateNewCourse(): void + calculateNewPosition(): void - updateVelocity(velocity_type): void - updateCourse(course_type): void -lookUpCourse(): course_type

TSS class:

TSS
- knox_rtiambModule: rtiamb_class * - newPtime, oldPTime, unitDistance: float + the_knox: KNOX * + ships_arr[]: Ship_Type + no_ships: unsigned int + shipAddedFlag: unsigned int + shipDeletedFlag: unsigned int
TSS() + set_the_knox(KNOX *): void + set_the_rtiamb(rtiamb_class *): void + setGhostShipName(char *): void + setGhostShipPosition(geocentric_pos_type) : void + createGhostShip(): void + getLocalGhostShip(): Ship_Type * - getShipName(): char * + addNewGhostToShipsArr(RTI::ObjectHandle): void + findObject(RTI::ObjectHandle): Ship_Type * + removeRemoteGhost(RTI::ObjectHandle): void + deleteLocalGhost(): void + deadReckonGhosts(): void

GPS Class:

GPS
- the_knox: KNOX *

GPS()
~GPS()
+ set_the_knox(KNOX *): void
+ show(): void
+ getVelocity(): velocity_type
+ getCourse(): course_type
+ getLAT(): formattedLatLongType
+ getLONG(): formattedLatLongType

EOT Class:

EOT
-eng_stat: eng_stat_type
-new_RPM: int
-the_knox: KNOX *
EOT()
~EOT()
+ return_engine_status(): eng_stat_type
+ setEngineStatus(eng_stat_type): void
+ set_the_knox(KNOX *): void
+ set_the_HydroShip(HydroShip *): void
- convertEngineStatusToRPM(eng_stat_type): int
- updateRPM(int): void // update RPM value in HydroShip

C.2.2 OTC FEDERATE CLASSES

TacticalConsole Class

TacticalConsole
- otcName: char *
- otcRank: char *
td: tacticalDisplay *
tss: TSS *
msgw: msgW *
geodetic: geodetic_pos_type

geocentric: geocentric_pos_type
TacticalConsole() ~TacticalConsole() + set_the_TSS(TSS*): void + set_the_TacticalDisplay(TacticalDisplay *): void + set_the_msgW(msgW *): void + setOtcName(char *): void + setOtcRank*char *): void + setPosition(geocentric_pos_type) : void + getotcName(): char * + getOtcRank(): char * + setEnviFile(char *): void + addShip(): void + removeShip(): void

sGUIPanel Class:

sGUIPanel
- shared: sGUI* - gps: GPS * - the_mc: messageCenter *
sGUIPanel() ~sGUIPanel() + set_the_sGUI(sGUI *): void + set_the_MC(messageCenter *): void + set_the_GPS(GPS *): void + initPanel(): pfuPanel * + handleEvents(): void + updateLatMsg(): void + updateLongMsg(): void + PanelControl(pfuWidget *): void

msgW Class:

msgW
tev: pfTextEnv * gstate: PfGeoState * msgDCS: pfDCS * msgRoot: pfGroup * sc: pfScene * gset: pfGeoSet * geode1: pfGeode *
msgW() ~msgW() +createScene(): pfScene()

APPENDIX – D

IMPLEMENTATION DETAILS

D.1 DEAD RECKONING ALGORITHM

Dead Reckoning (DR) is a technique used for reducing the network load in distributed interactive simulation by limiting the messages sent in network. *Tracking Sub-System (TSS)* class implements a dead reckoning algorithm using the "*players and ghosts*" paradigm [GLKP94]. Every player holds a list of others (ghost list) in the environment. Normally, if every player (e.g. ship) makes a movement in every time step, each must inform other players by sending messages (e.g. position message). Using dead reckoning, sender sends the information of its position, velocity, and acceleration, only when there is a change in its state. In this algorithm, sender decides to send the information if the difference between its real position and the estimated position calculated in its ghost exceeds an acceptable error amount (called threshold). In addition to the message sent when the error exceeds the threshold a message is sent according to a given period whatever error is. This periodic message is called *heartbeat*.

The dead reckoning algorithm implemented in NSTMSS has some differences due to the RTI and its architecture:

- The dead reckoning algorithm described in previous paragraph is developed for systems using message passing mechanism (e.g. DIS protocol)

- Heartbeat message is ignored because:
  - One of its two basic functions is to put an upper bound to errors caused by lost update messages. In NSTMSS architecture; all message delivery service is *reliable*.
  - Second is to give information about the environment to a newcomer player. In NSTMSS architecture, when a new player (federate) is joined to environment, then information is sent to the interested federates by the help of RTI.

- The position message is only sent when there is a state change in velocity and heading of the ship. Due to the absence of natural forces (e.g. wind, wave, and current), the calculation of player's real position and the calculation of its estimated position figured out in its ghost is the same and threshold is always zero.

D.2 CALCULATION OF A TRUE BEARING OF A RADAR ECHO

The basic characteristic of typical navigation radar used in tactical maneuvering is the ability to calculate the distance and true bearing of an echo. Although the calculation of the distance between an echo and our ship is simple, the calculation of true bearing is somewhat complex.

WHAT IS TRUE BEARING?

True bearing is the angle measured between the earth magnetic North and the radar echo. Relative bearing is the angle measured between the ship's bow and the radar echo.

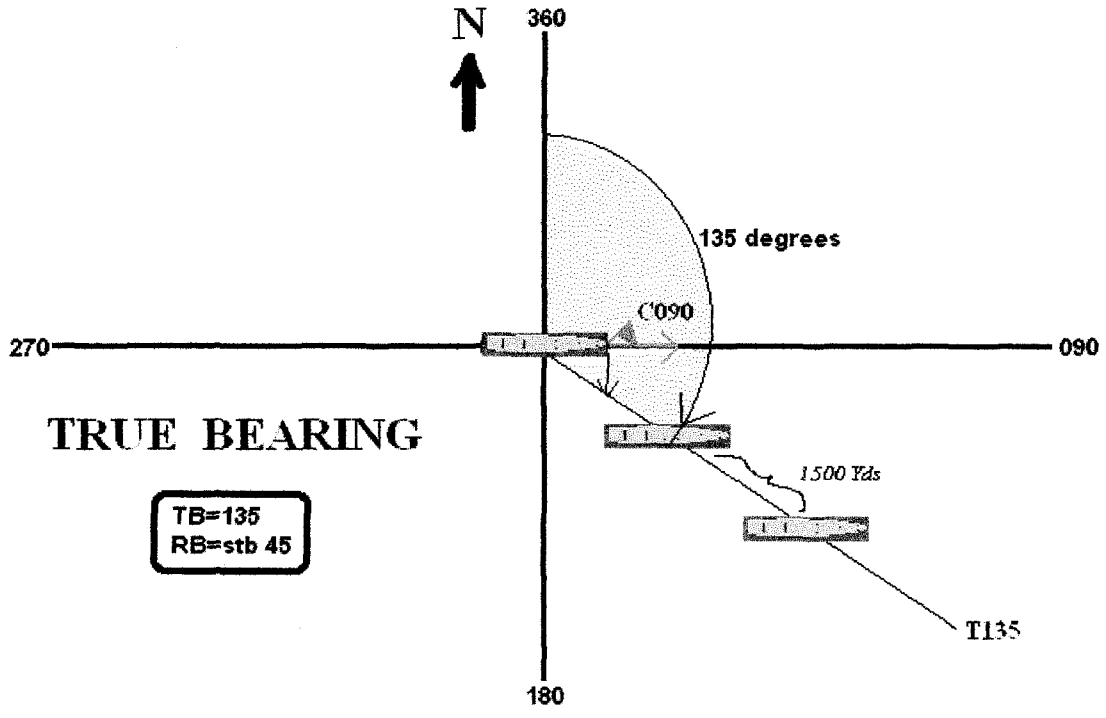


Figure D.1 True Bearing

ALGORITHM:

We know the value of position of our ship and the radar echo in terms of geocentric coordinate system (XYZ coordinates). TSS holds a list of ghosts in the environment. The Figure D.2 explains the mathematical model employed. The pseudo code of calculating the true bearing of an echo follows:

```
float calculateTrueBearing(*myShip, *selectedShip)
{
      // Get the positions
      x0=myShip-.>position.x;
      y0=myShip->position.y;
      x1=selectedShip->position.x;
      y1=selectedShip->position.y;

      // Calculate delta x and delta y and take their absolutes
      dx=abs(x1-x0);
      dy=abs(y1-y0);

      // check the regions and calculate alpha and bearing
      // positions are the same do it for avoiding zero division
erro
      if (x0==x1) && (y0==y1) bear=0.0f;

      //Region-I
      else if (x0<=x1)&&(y0<y1) then bear=arctan(dx/dy);

      //Region-II
      else if (x0>x1)&&(y0<=y1) then bear=3/2 *PI+arctan(dy/dx);

      //Region-III
      else if (x0>=x1)&&(y0>y1) then bear=PI+arctan(dx/dy);

      //Region-IV
      else if (x0<x1)&&(y0>=y1) then bear=PI/2+arctan(dx/dy);

      // Convert radian to degrees
      bear=bear*180/PI;

      return bear;
}.
```

For mathematical calculations (e.g. fabs, atanf), the ANSI C++ math library is used. Figure D.2 explains the pseudo code.

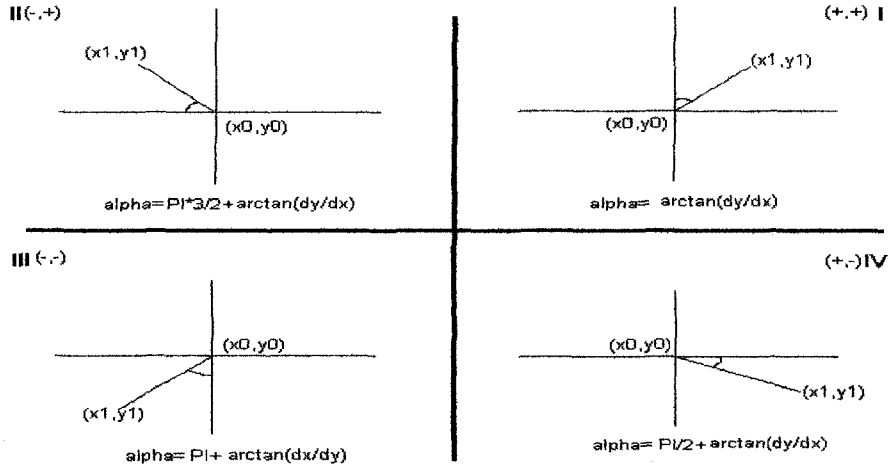


Figure D.2 Mathematical Model

D.3 EnviFd COMMAND LINE INTERFACE

Command line interface is implemented for OTC federate, KNOX federate, and Environment federate.

OTC Federate is known by OTC name. Usage:

```
USAGE:
 Process Name [rank] [name]

DEFAULTS:
 rank="Liet." Name="Okan_Topcu"

EXAMPLE:
 Virtue%>otc Liet Ahmet_Deniz
```

KNOX Federate is known by its ship name in the federation. Usage:

```
USAGE:
 Process Name [Ship Name]

DEFAULTS:
 Ship Name="TCG_PROTOTYPE"

EXAMPLE:
 Virtue%>knox TCG_MUAVENET
```

Environment Federate is known as "Environment" in the federation. The simulation loop consists of a command line interface to switch the fog type and time of the day interactively.

```
$ EnviFd>>h
$ ==========================
$ Enter:
$     " 't' -> To set Time of Day"
$     " 'f' -> To set fog status"
$     " 'h' -> To get here"
$     " 'q' -> To quit"
$ ==========================
```

For setting the fog status, type 'f' on the keyboard. The screen will display:

```
$ EnviFd>>f
$ Enter Fog Status (0-No Fog, 1-Some Fog): 1
$ Fog status is set
$ EnviFd>>
```


T.C. YÜKSEKÖĞRETİM KURULU
DOKÜMANTASYON MERKEZİ

APPENDIX – E

PRACTICAL MATTERS

E.1 FEDERATION DEVELOPMENT AND EXECUTION PROCESS (FEDEP) ISSUES

FEDEP is based on "waterfall with feedback model", so it relies on a well-scoped sequence of phases. The end product is coming at the end of implementations. FEDEP also assumes that SOMs of the federates are prepared beforehand. Those are the weaknesses of the present FEDEP model. They are discussed in chapter 6.

E.2 RUN TIME INFRASTRUCTURE (RTI) ISSUES

SERVICES INVOKED:

The following RTI management services are used (the detail of which federate uses which services is given in table 4.6/4.7/4.8):

- Federation Management Services: *Create federation execution, destroy federation execution, join federation execution, resign federation execution* services are used. The save/restore capability of RTI has not been tested.

- Declaration Management Services: All services described in the HLA interface specification are used.

- Object Management Services: All services are used except change *attribute transportation type, change interaction transportation type, attributes in scope, and attributes out of scope.* These are not used because a reliable transport scheme has been selected.

- Ownership Management Services: This service is not used at all in NSTMSS federation. All attributes in the federation are owned statically.

- Time Management Services: Although the federates are not time regulating and not time constrained, use of time management services allows RTI to be informed about the value of current federate time. The federation management processes will use the RTI time management schemes.

- Data Distribution Management Services: *Create/modify/delete region, subscribe interaction class with region, unsubscribe interaction class with region, send interaction with region* services are used.

PERFORMANCE:

Although this thesis does not test the performance of RTI implementation at hand, some points should be emphasized:

- The current RTI-version (RTI 1.3 release 6) requires the RTI-tick mechanism in order to continuously keep track of the current distributed simulation's state. This can diminish the scalability of real-time federations; because you cannot estimate the ticking time beforehand.

- The federate that tries to join the federation is locked while reading the FED file, if FEDEX has not heard from the previously joined federates for a long time.

E.3 FEDERATION EXECUTION DETAILS (FED) FILE ISSUES

The FED file is a text file representation of the Federation Object Model (FOM) which is read by the RTI and the federates. The RTI uses the FED file to assign handles to objects. Therefore it is important that federates within a federation have consistent FED files. The important issues about FED file are that:

- FED file must be located in *RTI_installation_home/config* folder of local host of each federate.

- As of the RTI1.3 release, names appearing in the FED file are case insensitive, so class "Ship" could be specified "ship", "SHIP", or "ShiP". In general, the case of the names in the FOM, Federation Execution Data, and federate source code should be considered as case-sensitive to ensure interoperability with all RTIs.

- While updating or editing FED file, *rtiexec* process should be terminated.

- Object classes and interaction classes can be constructed hierarchically. The following phrase is being extracted from *NSTMSS.fed* file.

```
(class Communication reliable receive
      (parameter Header)
      (parameter Subject)
      (parameter Body)
      (class TaskOrd reliable receive)
      (class WeatherRep reliable receive)
      (class FormMsg reliable receive)
 )
```

- The class and attribute names would appear in the FED file. As usual, the federates which want to publish or subscribe to a class, must get the

class handle with `RTI::getInteractionClassHandle(class name)` call. For the inherited classes (e.g. TaskOrd class), class name should be constructed as "*base_class.derived_class*". For example:

```
char* const
rtiamb_class::taskOrdStr="Communication.TaskOrd";
RTI::InteractionClassHandle taskOrdClassHandle=rtiamb->
getInteractionClassHandle(taskOrdStr);
```

- Interaction and object classes that are inherited do not inherit the routing space of the main class. Therefore, it is necessary to state the routing space for each derived class. Otherwise, you can get errors such as "*Invalid Region Context, Region space does not match interaction space*".

E.4 C++ ISSUES

Although the listed errors are well known for programmers, it is worth to review the most suffering compiler and run-time errors that I have encountered:

- Don't forget to initialize the static class members before you use them in the function codes. Otherwise, "*Unresolvable symbol in <executive name>*" error occurs in run-time.

- If you get "*segmentation fault*" error in compile-time. Segmentation fault happens during memory allocation (e.g. "new" calls), check out the pointers in your program.

- If you get "*Bus error: core dump*" error in run-time, check your Performer code, especially *pfScene* code. You do the things wrong with the graphics pipe (graphics card).

E.5 A FEW WORDS ABOUT CASE TOOL- Rose

Rational Rose is used for UML documentation in analysis and design of federates.

- Rational Rose does not include requirements analysis tools. For requirement specification, another tool should be used.

- The diagrams constructed in Rose, cannot be imported into a well-known format (e.g. *jpeg, gif, rgb,* etc.). Therefore, you cannot automatically generate reports. After completing a UML diagram, you should copy and paste the diagram into your document by using clipboard.

- Rose can automatically generate code from the UML diagrams. Although the generated code is poor, it was valuable to see the mapping of the UML diagrams into a programming language. The generated codes are class interfaces or definitions. Class diagrams are directly mapped into class prototypes. For example, when a one-to-one aggregation of two classes (call A and B) is mapped into C++ language or Java language, the produced code will likely be:

C++ Mapping

```
Class A
{
private:
  B *the_B;
Public:
  Set_the_B(B*);
  ---
};
```

```
Class B
{
private:
  A *the_A;
Public:
  Set_the_A(A*);
  ---
};
```

Java Mapping

```
Public Class A
{
 ---
public B the_B;
 ---
};
```

```
Public Class B
{
---
public A the_A;
---
};
```

APPENDIX – F

A NOTATION FOR FEDERATION DESIGN

F.1 MOTIVATION

All objects and interactions managed by a federate, and visible outside the federate, are described according to the standard Object Model Template (OMT). This common template facilitates understanding and comparisons of different federates and federations, and provides the format for a contract between members of a federation on the types of objects and interactions that will be supported across multiple interoperating simulations [DoD98c].

The primary objective of the HLA OMT is to facilitate interoperability among simulations and reuse of simulation components.

HLA object models are being documented using OMT components, which represent information about classes of objects, their attributes and their interactions in a *tabular way*. The template for the core of an HLA object model shall use a tabular format and still consist of the following components:

- *Object model identification table* associates important identifying information with the HLA object model
- *Object class structure table* records the namespace of all simulation/federation object classes and describes their class-subclass relationships

- *Interaction class structure table* records the namespace of all simulation/federation interaction classes and describes their class-subclass relationships
- *Attribute table* specifies features of object attributes in a simulation/federation
- *Parameter table*: specifies features of interaction parameters in a simulation/federation
- *Routing space table*: specifies routing spaces for object attributes and interactions in a federation
- *FOM/SOM lexicon* defines all of the terms used in the tables

HLA requires that these components shall be documented in the form of *tables*.

After briefly reviewing HLA OMT, now let's look into Federation Development and Execution Process (FEDEP). The FEDEP describes the activities involved in HLA federate and federation development and execution [DMSO98c]. The FEDEP does not specify a single way of constructing and executing an HLA federation, but rather identifies the key steps in federation development and execution [SWRL98]. It has five primary process stages: Requirements Definition, Conceptual Analysis, Federation Design, Federation Integration and Test, Federation Execution and Analysis.

After defining the federation requirements and scenario, we design and develop the federation. The primary purpose of Federation Design is to establish the membership of the federation, and to construct both a design framework and a formalized, lifecycle systems engineering approach for developing and implementing the federation. The federation design phase consists of such activities as:

- To assess the possibility of reusing an existing FOM that has been previously developed for different but similar applications.

- To determine the federates that will participate in federation, and to determine the suitability of existing SOMs.
- To build a formalized approach to federation development.
- To plan time management policies.

The purpose of the Federation Development phase is to develop the FOM, and to negotiate other products and resources that will be common across the federation. The federation development phase consists of:

- Federation Execution Data (FED) file generation
- To document the design architecture by constructing FOM according to OMT.

At the end of these phases, we document the federation object models by using OMT components in the tables in a tabular view.

To read and interpret data in these tables to understand the architecture of federation is a little bit hard. There is a need to depict the federation design graphically, which will provide more understandability and readability over complex and large-scale federations.

F.2 NOTATION

This graphical representation form shall not take the place of tabular description of HLA OMT, but shall enhance the understandability and readability of descriptions.

F.2.1 ARCHITECTURAL LEVEL FEDERATION DESIGN

All applications participating in a federation are called *Federates*. They can represent live entities or other facilities such as data collectors, and federation management federates.

A federate is represented an oval-like shape; the name of the federate being printed inside. Note that the federate names are not necessarily the run time names of the federate, which is identified by RTI.

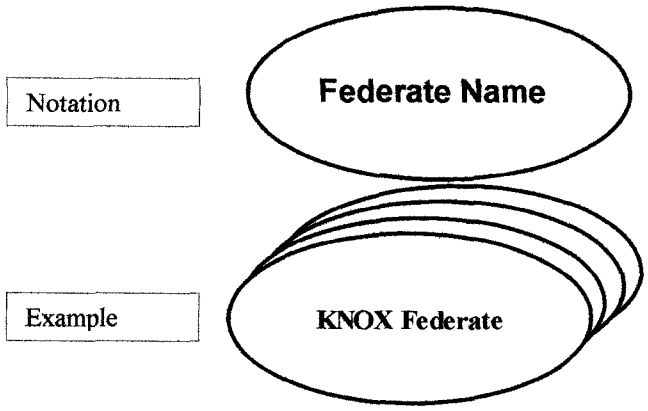


Figure F.1 Federate Notation

The recurring oval shapes describe *multiplicity* information that specifies the number of federates that participates in federation. The Table below summarizes the most common multiplicity values:

Table F.1 Multiplicity

Symbol	Meaning
1	One and only one
M	M (natural positive integers)
*	From 0 to any positive integer numbers
M..N	From M to N (positive integers)
0..N	From 0 to N (positive integers)

Processes that are not in federation, but are being used in construction of federation (i.e. RtiExec, FedExec, or only RTI), are presented by a rectangle, printed with its name.

Federates also can be logically grouped together to increase the abstraction level, so the readability. We can call these groups: *federate groups*. By using

this graphical representation, we can easily depict the architectural overview of federation in terms of federate or federate groups according to the abstraction level.

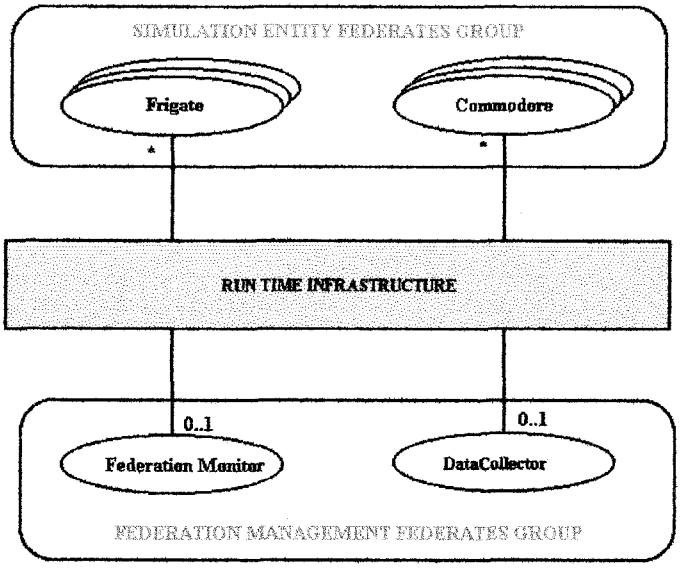


Figure F.2 Architectural Federation Design Example

Example: In scenario, there are many frigates and commodores who command the group of these maritime vessels.

The diagram in the Figure F.2 represents the architectural design of the federation. It gives an overview understanding of the federation. Frigate federate and Commodore federate represent real life entities and grouped together to form "simulation entity federates", other federates, federation monitor and data collector are grouped together as "federation management federates". The multiplicity is also being depicted in the architectural overview of the federation.

A federate group is represented by a rounded rectangle. Of course the use of colors and visual attributes are up to the designer. The federates are linked to RTI. *Links* represent bi-directional communication between federates and processes, especially between RTI and federates. Some external (not in

federation) components can also be added into diagram such as databases by linking them into federates.

F.2.2 HLA CLASSES

Differences between HLA and OOAD principles and concepts appear at the individual object level. In the OOAD literature, objects are defined as software encapsulations of data and operations (or methods). In the HLA, objects are defined entirely by the identifying characteristics (attributes) that are exchanged between federates during execution, with the behaviors and operations that affect the values of HLA object attributes kept resident in the federates. OOAD objects can be either tangible or conceptual, while HLA objects are normally intended to represent real-world entities (e.g., tank, aircraft, etc.).

HLA has two types of class: *Object-Class* and *Interaction-Class*. Object classes are comprised of attributes. Object classes generally describe the physical things that are going to be simulated (e.g. tanks, ships etc.). An *Attribute* is a named portion of an object state (e.g. velocity, position etc.). Interaction classes are comprised of *parameters*. Interaction classes describe types of events. Interaction instances are specific events (communications, collisions etc.). The primary difference between objects and interactions is persistence. Objects persist, interactions do not.

OBJECT AND INTERACTION CLASSES:

A class is represented by a rectangle as compartmentalized. The first compartment contains the class name. The class name should facilitate the understanding of what the class is. The other compartment contains the class's attributes. A class always contains its name. For interaction classes, the tag "`<interaction>`" should follow the class name.

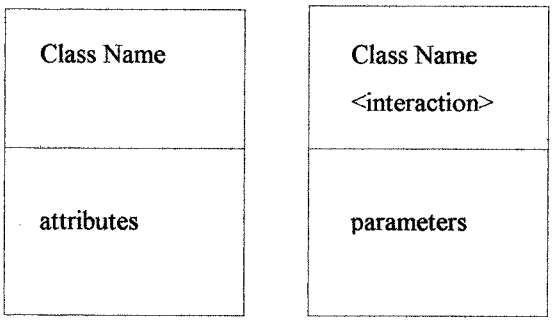


Figure F.3 Class notation

F.2.3 PUBLISH AND SUBSCRIBE DIAGRAMS

In federation design, it is essential to express the interest and capability of a federate about federation classes. The Publish/Subscribe pattern forms the basis of the model of communication proposed by HLA. The proposed diagram notation is useful for small-scale federations and it is developed for consistency in NSTMSS documentation (e.g. the documentation of this thesis and other researches related with NSTMSS).

General P/S Diagram of a federation depicts all the capability of existence federates in terms of objects and interactions that they produce or they are interested in. The *arc* from a federate to an object means that the federate has the capability to publish the object; on the other hand an arc from an object to a federate means that the federate is subscribed to the object or is interested in published object and its attributes.

Publish and subscribe diagrams can be divided in two sections to reflect the system in different views: *Class-Based Diagrams* and *Federate-Based Diagrams*.

Class Based Diagrams emphasize the P/S issues with a particular object and the federates. This type of diagrams is suitable for federate designers. For example, assume that a federate designer works to add a new federate into a federation. The first thing to do is to design the P/S interests with the existing federation objects and interactions. Figure F.4 shows an example of

a class-based diagram (refer to chapter 2 for more examples from NSTMSS).

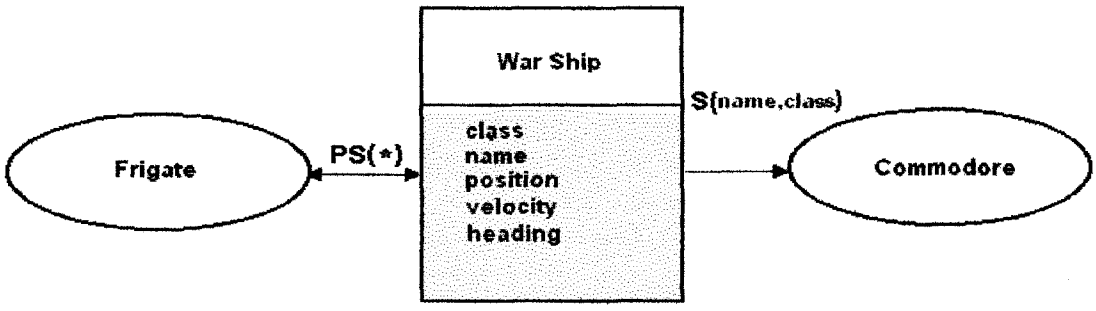


Figure F.4 Class-based P/S Diagram

The "P" stands for "publishes", "S" stands for "subscribes", and "PS" stands for "both publishes and subscribes". You can publish and subscribe in attribute level for object-classes. In other words, it is possible for a federate to publish or to subscribe a subset of the available attributes for a given class. For example, *Commodore* federate subscribes only the "*name*" and "*class*" attributes of "*War Ship*" class. This means that Commodore federate will receive only the *class and name* values when they are updated by *Frigate* federate. "*" means "all attributes". For example, *Frigate* federate has the capability both to subscribe and to publish the all attributes of War Ship class. For interaction classes, it is not possible to specify which parameters in an interaction will be published or subscribed. Interactions are produced as "all or nothing".

It is also possible to make hierarchies between classes. HLA object model only allows inheritance relation. At class based P/S diagrams, it is possible to represent the hierarchy. The figure F.5 gives an example. Here, `TaskOrd`, `WeatherRep`, and `FormMsg` interaction classes are inherited from the base class Communication. Example depicts the hierarchy and federates' interests.

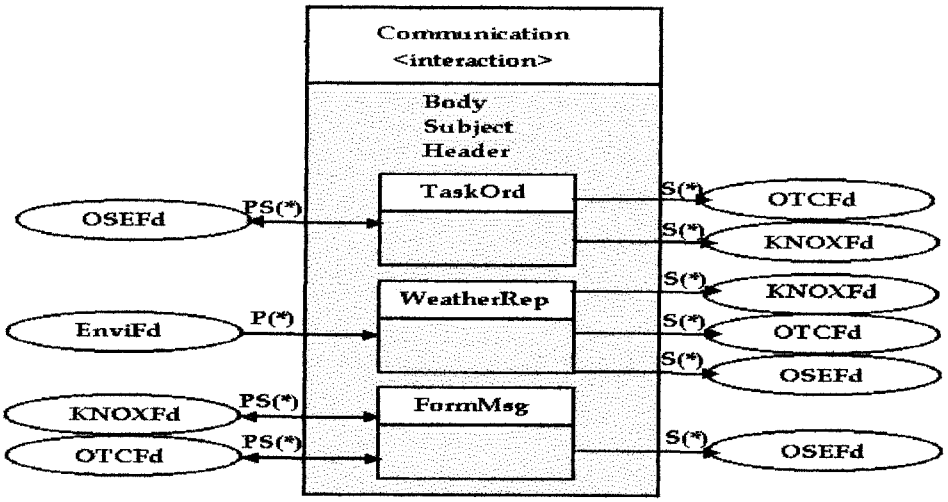


Figure F.5 P/S Diagram Example with Inherited Classes

The attributes of base class are promoted to the sub-classes. For example, When a federate subscribes to a derived class, say `TaskOrd`, it receives the `body`, `subject`, and `header` attributes such as it is subscribed to base class.

Federate-Based Diagrams depicts the capability and interest of a particular federate. It can be used when reviewing the existing federates for re-use. You can easily check the classes that the federate are capable to create or to use.

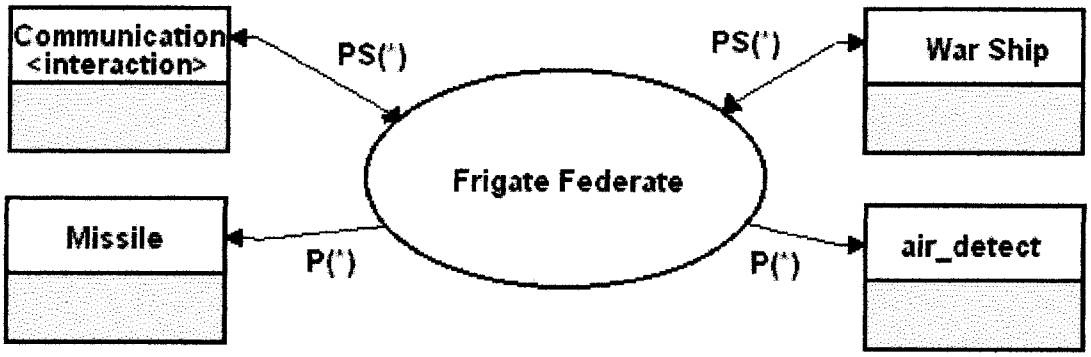


Figure F.6 Federate-Based P/S Diagram Example

APPENDIX – G

OBJECT MODEL TEMPLATE FOR NSTMSS

The HLA Object Model Template (OMT) provides a template for documenting HLA relevant information about classes of simulation or federation objects and their interactions [DoD98c]. This common template facilitates understanding and comparisons of different simulations and provides the format for a contract between members of a federation on the types of objects and interactions that will be supported across multiple interoperating simulations.

G.1 FEDERATION OBJECT MODELS (FOM)

During development of an HLA federation, it is critical that all federation members achieve a common understanding to the nature or character of all required communications among participating federates. The primary purpose of an HLA FOM is to provide a specification for data exchange among federates in a common, standardized format. The content of this data includes an enumeration of all object and interaction classes pertinent to the federation, along with a specification of the attributes or parameters that characterize these classes. Taken together, the individual components of an HLA FOM establish the "information model contract" that is necessary (but not sufficient) to achieve interoperability among the federates.

G.2 SIMULATION OBJECT MODELS (SOM)

An HLA SOM is a specification of the intrinsic capabilities that an individual simulation could provide to HLA federations. The standard format in which SOMs are expressed facilitates determination of the suitability of simulation systems for participation in a federation.

The HLA OMT formats are generally applicable to either FOMs or SOMs. Thus, SOMs are also characterized in terms of their objects, attributes, interactions, and parameters. The primary benefit from the common utilization of the OMT formats for FOMs and SOMs is that it provides a common frame of reference for describing object models in the HLA community. In some cases, this commonality may even allow SOM components to be integrated as "piece parts" in a FOM, facilitating rapid FOM construction.

G.3 HLA OMT COMPONENTS

HLA object models are composed of a group of interrelated components specifying information about classes of objects, their attributes, and their interactions. While the information content of these components can be represented in many different ways, the HLA requires that these components shall be documented in the form of tables. The template for the core of an HLA object model shall use a tabular format and shall consist of the following components:

- *Object model identification table*: to associate important identifying information with the HLA object model

- *Object class structure table*: to record the namespace of all simulation/federation object classes and to describe their class-subclass relationships

- *Interaction class structure table*: to record the namespace of all simulation/federation interaction classes and to describe their class-subclass relationships

- *Attribute table*: to specify features of object attributes in a simulation/federation

- *Parameter table*: to specify features of interaction parameters in a simulation/federation

- *Routing space table*: to specify routing spaces for object attributes and interactions in a federation

- *FOM/SOM lexicon*: to define all of the terms used in the tables

Both federations and individual simulations (federates) shall use all seven of the core OMT components when providing an HLA object model, although, in some cases, certain tables may be empty. However, all HLA object models shall contain at least one object class or one interaction class.

The routing space table will always be empty for SOMs, and may be empty for a FOM if no data distribution management (DDM) services are being used within the federation.

The FOM/SOM lexicon is essential to ensure that the semantics of the terms used in an HLA object model are understood and documented.

G.4 NSTMSS FOM DOCUMENTATION

At this section, NSTMSS FOM documentation is presented with explanations of tables.

G.4.1 OBJECT MODEL IDENTIFICATION TABLE

HLA object models provide information that enables inferences to be drawn regarding the reuse potential of individual federates for new applications. Reuse can also occur at the level of the object model itself. An existing FOM may provide a foundation for the development of new FOMs, or applicable components of existing SOMs may be merged to form new FOMs. In either case, to expedite reuse, it is important to include a minimum but sufficient degree of meta-level information in the object model description. For instance, when federation developers wish to pose detailed questions about how a federate or federation was constructed, including point-of-contact (POC) information within an HLA object model is extremely important.

Table G.1 Object Model Identification Table

OBJECT MODEL IDENTIFICATION TABLE	
Category	**Information**
Name	NSTMSS FOM
Version	1.1
Date	20.12.99
Purpose	Modeling the tactical maneuvering simulation system in a distributed interactive simulation environment using HLA concept
Application Domain	Naval Simulation Systems
Sponsor	MASLAB
POC	Liet. Okan Topcu
POC Organization	MASLAB/METU
POC Telephone	-
POC Email	otopcu@ceng.metu.edu.tr

G.4.2 OBJECT CLASS STRUCTURE TABLE

The object class structure of an HLA object model shall be defined by a set of relations among classes of objects from the simulation or federation domain. An HLA class structure shall be defined in terms of hierarchical relationships among classes of objects. Immediate class-to-subclass relationships shall be represented by the inclusion of the sub class names in the column to the right of the column for the base class in the object class structure table. HLA object model class hierarchies shall be represented by single inheritance (no multiple inheritance), although flat structures (with no subclasses) may also be used. In general, federates participating in a federation execution may subscribe to object classes at any level of the class hierarchy. By subscribing to all attributes of a specified object class, a federate is ensured of receiving all attribute value updates of attributes defined for that class and all of its superclasses for all instances of that class and all instances of its subclasses.

"P", "S", and "N" stands for *publishable, subscribable*, and *neither publishable or subscribable*.

Table G.2 Object Class Structure Table

OBJECT CLASS STRUCTURE TABLE			
Vehicle (N)	Surface Vehicle (N)	Ship (PS)	
Environment (PS)			
OTC (PS)			

G.4.3 INTERACTION CLASS STRUCTURE TABLE

Interactions shall be specified in the interaction class structure table of HLA object models in terms of their class-subclass relationships, in much the same way that objects are described in the object class structure table.

Four basic categories shall be used to indicate capabilities with respect to a given type of interaction:

- *initiates (I)*: indicates that a federate is currently capable of initiating and sending interactions of the given type
- *senses (S)*: indicates that a federate is currently capable of subscribing to the interaction and utilizing the interaction information, without necessarily being able to affect the appropriate changes to affected objects
- *reacts (R)*: indicates that a federate is currently capable of subscribing and properly reacting to interactions of the type specified by affecting the appropriate changes to any owned attributes of affected objects
- *neither initiates, senses, nor reacts* (N): indicates that a federate is not currently capable of initiating, sensing, or reacting to this interaction class.

Table G.3 Interaction Class Structure Table

INTERACTION CLASS STRUCTURE TABLE	
Communication (N)	TaskOrd (S)
	WeatherRep (S)
	FormMsg (IS)

G.4.4 ATTRIBUTE TABLE

Each class of simulation domain objects shall be characterized by a fixed set of attribute types. Table G.4 depicts the attribute table.

The *Cardinality* column shall be used to record the size of an array or sequence. A designation of *1+* in this column shall allow for unbounded sequences. The Units column shall designate the units (e.g., m, km, kg) used for each attribute whenever such units exist. The *Units, Resolution, Accuracy,* and *Accuracy Condition* columns are not applicable if the data type for an attribute is either enumerated or both complex and heterogeneous. The Accuracy column is intended to capture the maximum

deviation of the attribute value from its intended value in the federate or federation. The Update Type and Update Condition columns shall record the update policies for an attribute. The update type may be specified as *static*, *periodic*, or *conditional*. The Transferable/Acceptable (T/A) column is about ownership management. The Updateable/Reflectable (U/R) column of the attribute table shall be used to identify the current capabilities of a federate with respect to attribute updating and reflection. The Routing Space column shall record the association of an object attribute with a routing space if a federation is using DDM services.

G.4.5 PARAMETER TABLE

Most interaction classes will be characterized according to a list of one or more interaction parameters. Interaction parameters shall be used to associate relevant and useful information with classes of interactions. The parameter table of an HLA object model is designed to provide descriptive information about all interaction parameters represented in a federation. The explanation of the fields in the parameter table is the same with those in the attribute table. The table G.5 presents the characteristics of the parameters used in the federation.

G.4.6 ENUMERATED DATATYPE TABLE

The enumerated datatypes used in the federation are showed in Figure G.6.

Table G.4 Attribute Table

ATTRIBUTE TABLE

Object	Attr.	Data Type	Cardinality	Units	Resol.	Accuracy	Acc. Cond.	Update Type	Update Cond.	T/A	U/R	Rout. Space
Ship	name	string	1	N/A	N/A	perfect	always	conditional	When attr. is set	N	UR	Default
	position	Geocentric_postype	1	N/A	N/A	N/A	N/A	conditional	When heading or vel.	N	UR	Default
	heading	Course_type	1	Degree	N/A	N/A	N/A	conditional	When changed	N	UR	default
	velocity	Velocity_type	1	M/sec	N/A	N/A	N/A	conditional	When changed	N	UR	default
	fileName	string	1	N/A	N/A	perfect	always	conditional	When set	N	UR	default
Environment	TOD	TODType	1	N/A	N/A	N/A	N/A	conditional	when changed	N	UR	Default
	Fog Status	FogStatusType	1	N/A	N/A	N/A	N/A	Conditional	When changed	N	UR	default

Table G.5 Attribute Table (cont.)

ATTRIBUTE TABLE											
Object	**Attr.**	**Data Type**	**Cardinality**	**Units**	**Resol.**	**Accuracy**	**Acc. Cond.**	**Update Type**	**Update Cond.**	**T/A**	**U/R**	**Rout. Space**
Environment (Cont.)	LLC	Geocentric_postype	1	N/A	N/A	N/A	N/A	Conditional	When set	N	UR	Default
	URC	Geocentric_postype	1	N/A	N/A	N/A	N/A	conditional	When set	N	UR	Default
OTC	name	string	1	N/A	N/A	perfect	always	conditional	When set	N	UR	default
	Rank	string	1	N/A	N/A	perfect	Always	conditional	when set	N	UR	Default

Table G.6 Parameter Table

PARAMETER TABLE

interaction	Parameter	Data Type	Cardinality	Units	Resol.	Accuracy	Acc. Cond.	Rout. Space
Communication	header	headerType	1	N/A	N/A	N/A	N/A	Tactical Communication
	subject	string	1	N/A	N/A	N/A	N/A	Tactical Communication
	body	BodyType	1	N/A	N/A	N/A	N/A	Tactical Communication
Comm.taskOrd	-	-	-	-	-	-	-	Tactical Communication (HF)
Comm.WeatherRep	-	-	-	-	-	-	-	Tactical Communication (VHF)
Comm.FormMsg	-	-	-	-	-	-	-	Tactical communicat ion (UHF)

Table G.7 Enumerated Datatype Table

ENUMERATED DATATYPE TABLE		
Identifier	**Enumerator**	**Representation**
TODType	DAY	0
	EVENING	1
fogStatusType	NO_FOG	0
	FOG1	1

G.4.7 COMPLEX DATATYPE TABLE

Complex datatype identifiers shall match a datatype entry from the attribute table, the parameter table, or from the complex datatype table itself. The complete set of basetypes (e.g. float, char, short etc.) is presented in HLA OMT Specification [DoD98c]

Table G.8 Complex Datatypes

COMPLEX DATATYPE TABLE							
type	**Field name**	**Datatype**	**Cardinality**	**Units**	**Resolution**	**Accuracy**	**Acc.Cond**
Geocentric_pos_type	X	float	1	m	N/A	Perfect	Always
	Y	float	1	m	N/A	Perfect	Always
	z	float	1	m	N/A	Perfect	Always
Course_type	-		1	Degree	N/A	Perfect	Always
Velocity_type	-	Float	1	Knots	N/A	Perfect	Always
HeaderType	FmText	Char	15	N/A	N/A	Perfect	Always
	ToText	Char	15	N/A	N/A	Perfect	Always

	CcText	Char	15	N/A	N/A	Perfect	Always
BodyType	bd1Text	Char	15	N/A	N/A	Perfect	Always
	bd2Text	Char	15	N/A	N/A	Perfect	Always
	bd3text	Char	15	N/A	N/A	Perfect	Always

G.4.8 ROUTING SPACE TABLE

Routing spaces are the most fundamental DDM concept. A routing space is a multidimensional coordinate system through which federates either express an interest in receiving data or declare their intention to send data. During development of an HLA federation, it is critical that all federation members achieve a common understanding of DDM routing spaces and their semantics, and agree to a common set of routing space specifications. These agreements are necessary for federates to filter object attribute updates and interactions in a correct and consistent manner.

Table G.9 Routing Space Table

ROUTING SPACE TABLE					
Routing Space	**Dimension**	**Dimension Type**	**Dimension Range Set**	**Range/Set Units**	**Normalization Function**
Tactical Communication Space	Frequency Spectrum	Unsigned Short	300-3000	MHz	Linear

G.4.9 FOM/SOM LEXICON

If interoperability among simulations is to be achieved, it is necessary not only to specify the classes of data required by the templates but also to achieve a common understanding of the semantics of this data. The FOM/SOM lexicon provides a means for federations to document the

definitions of all terms utilized during construction of FOMs, and for individual federates to document the definitions of all terms provided in their SOMs.

Table G.10 Object Class Definitions

OBJECT CLASS DEFINITIONS	
Term	**Definition**
Ship	Ship object class represents any ship in real life. Ship object class holds the entity information (name, class, position etc.) of the ships in the environment.
Environment	Environment object class encapsulates the file name and the limits of the area, in which scenario takes place. This class informs the subscribed federates about environment status and changes (e.g. from a foggy day to a sunny day).
OTC	OTC object class is an abstraction of Officer in Tactical Command. OTC commands the Surface Task Group.

Table G.11 Interaction Class Definitions

INTERACTION CLASS DEFINITIONS	
Term	**Definition**
Communication	Tactical Message System (TMS) deals with all the communication messages between the commodores and captains. TMS describes the message types. Communication interaction class is the abstraction of these message types.
Communication.TaskOrd	Tasking Order message is used for describing the whole scenario. It is generated by OSE and sent to OTC and to ships.
Communication.WeatherRep	Weather Report describes the weather condition in the scenario and it is sent by local meteorology station to participants

Communication.FormMsg	Formation Messages are used in order to construct formations between ships and OTC.

G.5 KNOX FEDERATE SOM DOCUMENTATION

Table G.12 KNOXFd Object Model Identification Table

OBJECT MODEL IDENTIFICATION TABLE	
Category	**Information**
Name	KNOX Federate (KNOXFd) SOM
Version	4.2
Date	20.12.99
Purpose	To model and simulate tactical maneuvering of a KNOX class frigate
Application Domain	Naval Simulation Systems
Sponsor	MASLAB
POC	Liet. Okan Topcu
POC Organization	MASLAB/METU
POC Telephone	-
POC Email	otopcu@ceng.metu.edu.tr

Table G.13 KNOXFd Object Class Structure Table

OBJECT CLASS STRUCTURE TABLE			
Vehicle (N)	Surface Vehicle (N)	Ship (PS)	
Environment (P)			

Table G.14 KNOXFd Interaction Class Structure Table

INTERACTION CLASS STRUCTURE TABLE	
Communication (N)	TaskOrd (S)
	WeatherRep (S)
	FormMsg (IS)

Table G.15 KNOXFd Complex Datatypes

COMPLEX DATATYPE TABLE							
type	**Field name**	**Datatype**	**Cardinality**	**Units**	**Resolution**	**Accuracy**	**Acc.Cond**
Geocentric_pos_type	x	float	1	m	N/A	Perfect	Always
	y	float	1	m	N/A	Perfect	Always
	z	float	1	m	N/A	Perfect	Always
Course_type	-		1	Degree	N/A	Perfect	Always
Velocity_type	-	Float	1	Knots	N/A	Perfect	Always
HeaderType	FmText	Char	15	N/A	N/A	Perfect	Always
	ToText	Char	15	N/A	N/A	Perfect	Always
	CcText	Char	15	N/A	N/A	Perfect	Always
BodyType	bd1Text	Char	15	N/A	N/A	Perfect	Always
	bd2Text	Char	15	N/A	N/A	Perfect	Always
	bd3text	Char	15	N/A	N/A	Perfect	Always

Table G.16 KNOXFd Attribute Table

ATTRIBUTE TABLE											
Object	**Attr.**	**Data Type**	**Cardinality**	**Units**	**Resol.**	**Accuracy**	**Acc. Cond.**	**Update Type**	**Update Cond.**	**T/A**	**U/R**	**Rout. Space**
Ship	name	string	1	N/A	N/A	perfect	always	conditional	When attr. Is set	N	UR	Default
	position	Geocentric_postype	1	N/A	N/A	N/A	N/A	conditional	When heading or vel. Changed	N	UR	Default
	heading	Course_type	1	Degree	N/A	N/A	N/A	conditional	When changed	N	UR	default
	velocity	Velocity_type	1	M/sec	N/A	N/A	N/A	conditional	When changed	N	UR	default
Environment	String	string	1	N/A	N/A	perfect	always	conditional	When set	N	R	default
	TOD	TODType	1	N/A	N/A	N/A	N/A	conditional	when changed	N	R	Default
	Fog Status	FogStatusTyp	1	N/A	N/A	N/A	N/A	Conditional	When changed	N	R	default

Table G.17 KNOXFd Parameter Table

PARAMETER TABLE

Interaction	Parameter	Data Type	Cardinality	Units	Resol.	Accuracy	Acc. Cond.	Rout. Space
Communication	header	headerType	1	N/A	N/A	N/A	N/A	Default
	subject	string	1	N/A	N/A	N/A	N/A	Default
	body	BodyType	1	N/A	N/A	N/A	N/A	Default
Comm.taskOrd	-	-	-	-	-	-	-	Tactical Communication (HF)
Comm.WeatherRep	-	-	-	-	-	-	-	Tactical Communication (VHF)
Comm.FormMsg	-	-	-	-	-	-	-	Tactical communicat ion (UHF)

Table G.18 KNOXFd Enumerated Datatype Table

ENUMERATED DATATYPE TABLE		
Identifier	**Enumerator**	**Representation**
TODType	DAY	0
	EVENING	1
fogStatusType	NO_FOG	0
	FOG1	1

Table G.19 KNOXFd Routing Space Table

ROUTING SPACE TABLE					
Routing Space	**Dimension**	**Dimension Type**	**Dimension Range Set**	**Range/Set Units**	**Normalization Function**
Tactical Communication Space	Frequency Spectrum	Unsigned Short	300-3000	MHz	Linear

Table G.20 KNOXFd Object Class Definitions

OBJECT CLASS DEFINITIONS	
Term	**Definition**
Ship	Ship object class represents any ship in real life. Ship object class holds the entity information (name, class, position etc.) of the ships in the environment.
Environment	Environment object class encapsulates the file name and the limits of the area, in which scenario takes place. This class informs the subscribed federates about environment status and changes (e.g. from a foggy day to a sunny day).

Table G.21 KNOXFd Interaction Class Definitions

INTERACTION CLASS DEFINITIONS	
Term	**Definition**
Communication	Tactical Message System (TMS) deals with all the communication messages between the commodores and captains. TMS describes the message types. Communication interaction class is the abstraction of these message types.
Communication.TaskOrd	Tasking Order message is used for describing the whole scenario. It is generated by OSE and sent to OTC and to ships.
Communication.WeatherRep	Weather Report describes the weather condition in the scenario and it is sent by local meteorology station to participants
Communication.FormMsg	Formation Messages are used in order to construct formations between ships and OTC.

G.5 OTC FEDERATE SOM DOCUMENTATION

Table G.22 OTCFd Object Model Identification Table

OBJECT MODEL IDENTIFICATION TABLE	
Category	**Information**
Name	OTC Federate (OTCFd) SOM
Version	3.0
Date	20.12.99
Purpose	To model and simulate tactical level simulation in naval operations

Application Domain	Naval Simulation Systems
Sponsor	MASLAB
POC	Liet. Okan Topcu
POC Organization	MASLAB/METU
POC Telephone	-
POC Email	otopcu@ceng.metu.edu.tr

Table G.23 OTCFd Object Class Structure Table

OBJECT CLASS STRUCTURE TABLE			
OTC(P)			
Environment (S)			

Table G.24 OTCFd Interaction Class Structure Table

INTERACTION CLASS STRUCTURE TABLE	
Communication (N)	TaskOrd (S)
	WeatherRep (S)
	FormMsg (IS)

Table G.25 OTCFd Complex Datatypes

COMPLEX DATATYPE TABLE							
type	**Field name**	**Datatype**	**Cardinality**	**Units**	**Resolution**	**Accuracy**	**Acc.Cond**
HeaderType	FmText	Char	15	N/A	N/A	Perfect	Always
	ToText	Char	15	N/A	N/A	Perfect	Always
	CcText	Char	15	N/A	N/A	Perfect	Always
BodyType	bd1Text	Char	15	N/A	N/A	Perfect	Always
	bd2Text	Char	15	N/A	N/A	Perfect	Always

	bd3text	Char	15	N/A	N/A	Perfect	Always

Table G.26 Enumerated Datatype Table

ENUMERATED DATATYPE TABLE		
Identifier	**Enumerator**	**Representation**
TODType	DAY	0
	EVENING	1
FogStatusType	NO_FOG	0
	FOG1	1

Table G.27 Routing Space Table

ROUTING SPACE TABLE					
Routing Space	**Dimension**	**Dimension Type**	**Dimension Range Set**	**Range/Set Units**	**Normalization Function**
Tactical Communication Space	Frequency Spectrum	Unsigned Short	300-3000	MHz	Linear

Table G.28 OTCFd Attribute Table

ATTRIBUTE TABLE

Object	Attr.	Data Type	Cardinality	Units	Resol.	Accuracy	Acc. Cond.	Update Type	Update Cond.	T/A	U/R	Rout. Space
OTC	name	string	1	N/A	N/A	perfect	always	conditional	When set	N	U	Default
	Rank	String	1	N/A	N/A	Perfect	always	conditional	When set	N	U	Default
	String	string	1	N/A	N/A	perfect	always	conditional	When set	N	R	default
Environment	TOD	TODType	1	N/A	N/A	N/A	N/A	conditional	when changed	N	R	Default
	Fog Status	FogStatusType	1	N/A	N/A	N/A	N/A	Conditional	When changed	N	R	default

Table G.29 OTCFd Parameter Table

PARAMETER TABLE

interaction	Parameter	Data Type	Cardinality	Units	Resol.	Accuracy	Acc. Cond.	Rout. Space
Communication	header	headerType	1	N/A	N/A	N/A	N/A	Default
	subject	string	1	N/A	N/A	N/A	N/A	Default
	body	BodyType	1	N/A	N/A	N/A	N/A	default
Comm.taskOrd	-	-	-	-	-	-	-	Tactical Communication (HF)
Comm.WeatherRep	-	-	-	-	-	-	-	Tactical Communication (VHF)
Comm.FormMsg	-	-	-	-	-	-	-	Tactical communicat

Table G.30 Object Class Definitions

OBJECT CLASS DEFINITIONS	
Term	**Definition**
OTC	OTC object class is an abstraction of Officer in Tactical Command. OTC commands the Surface Task.
Environment	Environment object class encapsulates the file name and the limits of the area, in which scenario takes place. This class informs the subscribed federates about environment status and changes (e.g. from a foggy day to a sunny day).

Table G.31 Interaction Class Definitions

INTERACTION CLASS DEFINITIONS	
Term	**Definition**
Communication	Tactical Message System (TMS) deals with all the communication messages between the commodores and captains. TMS describes the message types. Communication interaction class is the abstraction of these message types.
Communication.TaskOrd	Tasking Order message is used for describing the whole scenario. It is generated by OSE and sent to OTC and to ships.
Communication.WeatherRep	Weather Report describes the weather condition in the scenario and it is sent by local meteorology station to participants
Communication.FormMsg	Formation Messages are used in order to construct formations between ships and OTC.

APPENDIX – H

GLOSSARY

A

Astern: At or toward the rear of a ship [Tr.: Pupa]

B

Bearing: Direction or position in relation to another point or to the points of a compass [Tr.: Kerteriz]

C

Cadet: Student in a military academy in training for service as an officer [Tr.: Askeri Öğrenci]

Commanding Officer (CO): Officer having command of a military ship [Tr.: Gemi Komutani]

Commodore: Officer in tactical command of a flotilla [Tr.: Komodor].

D

Deplacement: The volume of the ship under waterline [Tr.: Deplasman]

E

Engine Order Telegraph (EOT): The machine that relays the engine orders to the engine room [Tr.: Makine Telgrafi]

F

Federate: A member of a HLA federation; one simulation

Federation: A named set of interacting federates, a common federation object model, and supporting RTI, that are used as a whole to achieve some specific objective

Formation: The predefined order and position for each ship, which is joined into a surface task group [Tr.: Nizam].

Formation Steaming: The movement of ships in the surface task group for the sake of keeping their positions in the formation [Tr.: Nizam Seyri].

G

Gyrocompass: Compass using a rapidly spinning gyroscope to indicate the north [Tr.: Cayro Pusula]

H

Helm: Tiller, wheel, or entire steering apparatus of a ship [Tr.: Dumen]

I

Interaction: A non-persistent time tagged event generated by one federate and received by others

J

Junior Officer Of the Deck: Novice Officer of the Deck [Tr.: Vardiya Subayi]

N

Nautical Mile (nm): 1852m; 2000 yds [Tr.: Deniz Mili].

O

Officer in Tactical Command (OTC): The superior commanding officer of the STG; STGC; commodore [Tr.: Taktik Komutan].

Officer Of the Deck (OOD): The superior officer of the watch and responsible from navigation of the ship [Tr.: Vardiya Amiri].

Officer Scheduling the Exercise (OSE): Authority that defines the task, the plan; the chronology of the events of the naval operation [Tr.: Tatbikati Planlayan Makam].

P

Parameter: A named datum associated with each instance of a class of interactions

Port: Left side of a ship [Tr.: İskele]

R

Relative Bearing: Bearing relative to the ship course [Tr.: Nisbi Kerteriz]

Rudder: Movable piece of metal or similar material, usually attached vertically at the stern of a ship, and used in steering [Tr.: Dümen Yelpazesi]

Rules of the Road: International Regulations for Preventing Collisions at Sea. [Tr.: Denizde Çatişmayi Önleme Tüzüğü]

S

Starboard: Right side of a ship [Tr.: Sancak]

Stern: Rear part of a ship [Tr.: Kıç]

Surface Task Group: A naval surface force, which is composed two or more ships to achieve a specific task [Tr.: Suüstü Görev Grubu].

Surface Operation: A set of military movements, which is carried out by naval surface ships to achieve a specific task [Tr.: Suüstü Harekati]

Surface Task Group Commander: The superior commanding officer of the STG [Tr.: Suüstü Görev Grubu Komutani]

T

True Bearing: Bearing according to the earth magnetic field [Tr.: Hakiki Kerteriz].

Turn: Moving around on or as on an axis of ships in the formation [Tr.: Dönüş].

Y

Yard: 91.5cm [Tr.: Yarda].

REFERENCES

[AEGIS98] Aegis Research Corporation, "*OMDT User's Guide Beta1.1*", Huntsville AL, 1998

[B62] Blagoveschcensky, S.N., "*Theory of Ship Motions*", Dover Publications Inc., Newyork, 1962

[BP97] Babeniew William E., Page Ernest H., "*The ALSP Joint Training Confederation A Case Study of Federation Testing*", Simulation interoperability Workshop, IST, Fall 1997

[CWZ99] Capps M., Watsen K., and Zyda M., "*Cyberspace and Moch Apple Pie: A Vision of the Future of Graphics and Virtual Environments*", Nov/Dec 99 issue of IEEE Computer Graphics and Applications (CGA99), 1999

[D98] Dahmann J., "*High Level Architecture for Simulation*", Defense Modeling and Simulation Office, 1998

[DMSO96] Defense Modeling and Simulation Office-DMSO, "*HLA Time Management Design Document v1.0*", 15 Aug 1996

[DMSO98a] Defense Modeling and Simulation Office-DMSO, "*HLA Glossary*", 1998

[DMSO98b] Defense Modeling and Simulation Office-DMSO, "*High Level Architecture Run-time Infrastructure Programmer's Guide v1.3r4*", Oct 1998

[DMSO98c] Defense Modeling and Simulation Office-DMSO, "*HLA Federation Development and Execution Process (FEDEP) Model, version 1.2*", 26 May 1998

[DoD98a] U.S. Department of Defense, "*Draft Standard for Modeling & Simulation High Level Architecture-Rules version 1.3*", IEEE P1516/D1, April 1998.

[DoD98b] U.S. Department of Defense, "*Draft Standard for Modeling & Simulation High Level Architecture-Federate Interface Specification*", IEEE P1516.1, February 1998

[DoD98c] U.S. Department of Defense "*Draft Standard for Modeling & Simulation High Level Architecture-Object Model Template*", IEEE P1516.2, February 1998

[DoD98d] U.S. Department of Defense "*HLA Federation Execution Planner's Workbook (FEPW) Version 1.3 Draft 7"*, 10 June 1998

[GLKP94] Grossweiler R., Laferriere R., Keller M.L., Pausch R., "*An Introductory Tutorial For Developing Multi-User Virtual Environments*", Presence, December,3 1994, pp.255-264

[GM88] Garvey R., Mondey P., "*SIMNET (SIMulation NETworking)*", BBN Systems and Technologies, Belleuve WA, July 1988

[MITRE95] the MITRE Corporation, "*ALSP Project 1994 Annual Report*", March 1995

[MM97] Muller Pierre-Alain, " *Instant UML*", Wrox Press Ltd., 1997

[MS65] Munro-Smith R., "*Notes and Examples in Naval Architecture*", Edword Arnold Publishers Ltd., London 1965

[NG95] Nobles J.A., Garrova J.F, **"***Design and Implementation of Real-Time Deployable three Dimensional Shiphandling Training Simulator"*, Master;s thesis, Naval Postgraduate School, Montery California, June 1995

[OMG99] Object Management Group, "OMG Unified Modeling Language Specification Version 1.3 Alpha R2", January 1999

[R98] Ralph M.Toms, "*New Efficient Procedures for Geodetic Coordinate Transformations* ", Simulation Interoperability Workshop, Simulation, 1998 spring

[RSC98] Rational Software Corporation, "*Introduction to Rational Rose98*", Lexington MA USA, 1998

[S97] Stroustrup B., "*The C++ Programming Language*", 3rd edition, Addison-Wesley, 1997

[SA99] Ship Analytics, "*Ship Analytics Products and Services Catalog*", Connecticut USA, 1999

[SGI97] Silicon Graphics Inc. "*IRIS Performer Getting Started Guide*" Document Number 007-3560-001, 1997

[SWRL98] Scrudder R., Waite W., Richardson M., Lutz R., "*Graphical Presentation of the FEDEP*", Simulation Interoperability Workshop, Fall 1998

[USCG95] US Department of Transportation United States Coast Guard "*Navigation Rules International-Intend*", 01 Oct 1995

[Z93] Zaestwitz S., "*NPSNET: Integration of Distributed Interactive Simulation (DIS) Protocol for Communication Architecture and Information Exchange*", Master's thesis, Naval Postgraduate School, Montery California, September 1993

[ZLPM92a] Zyda M., Locke J., Pratt D., Michael J., "*A DIS network library for UNIX and NPSNET*", 7th Workshop on Standards for the Interoperability of Defense Simulations, September1992

[ZLPM92b] Zyda M., Locke J., Pratt D., Michael J., "*Integrating SIMNET with NPSNET using a mix of Silicon Graphics and Sun Workstations*", 6th Workshop on Standards for the Interoperability of Defense Simulations, March 1992

[ZMPB94] Zyda M., Macedonia M., Pratt D., Barham P., Zaestwitz S., "*NPSNET: A Network Architecture for Large Scale Virtual Environments*", Presence Vol.3 No4, Fall 1994

[ZPFB93] Zyda M., Pratt D., Falby J., Baraham P., "*NPSNET and the Naval Postgraduate School Graphics and Video Laboratory*", Presence Vol.2 No3, 1993

SELECTED WWW SITES

[1] High Level Architecture http://hla.dmso.mil by DMSO

[2] DMSO HLA Example http://pavg.stanford.edu/rapide/examples/dmso/dmso-hla.html By Stanford Rapide Project

[3] DIS HLA transition http://www.mak.com By MAK Technologies

[4] OMDT http://www.aegisrc.com/Products/index.htm By Aegis Research

[5] OMDT User Guide http://www.tasc.com/projects/omdt/help/toc_table.html By Litton TASC

[6] A Tactical Communications Modeling Approach for Advanced Distributed Simulation http://www.ait.nrl.navy.mil/5585/Papers/TCMS.html By Naval Research Laboratory

[7] Full Mission Shiphandling Simulator http://www.shipanalytics.com By ShipAnalytics

[8] Naval Simulation System http://www.advtech543.saic.com/NSS_SAIC.HTM By SAIC

[9] Nautical and Naval Simulation http://www.stn-atlas.de/engl/e_sim/e_en.htm By STN ATLAS Elektronik GmbH

[10] NPSNET Research Group http://www-npsnet.cs.nps.navy.mil By Navy Post Graduate School

[11] Modeling & Simulation Information Analysis Center (MSIAC) http://www.msiac.dmso.mil by DMSO

[12] Conceptual Model of Mission Space (CMMS) Resource Center http://38.241.48.9 by DMSO

[13] NATO Modeling And Simulation Group http://www.rta.nato.int/msg.htm by NATO

[14] Modeling and Simulation Resource Repository (MSRR) http://www.mssr.dmso.mil by DMSO

T.C. YÜKSEKÖĞRETİM KURULU
DOKÜMANTASYON MERKEZİ