THE DEVELOPMENT OF A SCENARIO TRANSLATOR FOR DISTRIBUTED SIMULATIONS

THESIS

Heon-Gyu Park, Captain
Republic of Korea, Army

AFIT/GCS/ENG/96D-22

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/96D-22

THE DEVELOPMENT OF A SCENARIO TRANSLATOR FOR DISTRIBUTED SIMULATIONS

THESIS

Heon-Gyu Park, Captain
Republic of Korea, Army

AFIT/GCS/ENG/96D-22

DTIC QUALITY INSPECTED 2

Approved for public release; distribution unlimited

19970409 041

AFIT/GCS/ENG/96D-22

The Development of a Scenario Translator for Distributed Simulations

THESIS

Presented to the Faculty of the Graduate School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Systems

Heon-Gyu Park, B.S.
Captain, Republic of Korea, Army

December 1996

Approved for public release; distribution unlimited

Preface

I can do all things through Christ, which strengtheneth me. Pilippians 4:13

I would like to thank everyone who helped me to finish this thesis effort. My first gratitude goes to my advisor, Maj. Keith Shomper. His patience, enthusiasm, and understanding of my situation guided me through this thesis. He showed me the way every time I encountered problems. My thesis committee, Lt. Col. Martin Stytz and Dr. Hartrum, provided me their positive energy and constant faith in my abilities. Their fairness and encouragement kept me going through the program.

I owe a sincere thanks to the ModSAF, EADSIM, and BATTLESIM developers those who met with me via the network or telephone. They gave me correct answers whenever I asked about the simulation. I wish to thank the many people at ModSAF reflector and Teledyne Brown User Support Group who quenched my thirst for knowledge.

I'm also indebted to a number of people who graciously extended their patience and friendship during the time I was writing this thesis. My hearty thanks goes to Doug Blake who devoted his time and effort to reviewing this thesis.

I also appreciate the cheering and encouragement of the Korean Officers who showed their interest and concern for my effort, and I want to express gratitude to the Christians of my church who offered a lot of prayers for me.

I dedicate this thesis to my family in gratitude for their eternal support, faith and love. I don't know how to thank my wife, Hae-Suk, who endure patiently everything until I graduated and my son Ji-Won who missed lots of fun with me. Finally, I send all my honor round to God who guided me, and also will guide me through my life before I face him.

Heon-Gyu Park

Table of Contents

	Page
Preface	ii
Table of Contents	iii
List of Figures	vi
List of Tables	viii
List of Abbreviations	xii
Abstract	xiv
I. Introduction	1 - 1
1.1 Background	1 - 1
1.2 Problem Statement	1 - 3
1.3 Assumption	1 - 3
1.4 Approach	1 - 4
1.4.1 Determining Conversion Method	1 - 4
1.4.2 Mapping to Transitional Prototype	1 - 6
1.4.3 Remapping to Target Scenario	1 - 7
1.5 Thesis Overview	1 - 7
II. Background and Literature Review	2 - 1
2.1 Introduction	2 - 1
2.2 Historical Background	2 - 1
2.3 Military Simulation Technology	2 - 2
2.3.1 Distributed Interactive Simulation (DIS)	2 - 2
2.3.1.1 Concept	2 - 2
2.3.1.2 DIS Objectives	2 - 3
2.3.1.3 DIS PDUs	2 - 3
2.3.2 Advance Distributed Simulation	2 - 4
2.3.3 Computer Generated Forces	2 - 7
2.4 Distributed Simulations	2 - 8
2.4.1 Modular Semi-Automated Forces (ModSAF)	2 - 8
2.4.1.1 ModSAF Architecture	2 - 8
2.4.1.2 Scenario File Organization	2 - 10
2.4.2 Extended Air Defense Simulation (EADSIM)	2 - 11
2.4.2.1 EADSIM Architecture	2 - 12
2.4.2.2 Scenario File Organization	2 - 13
2.4.3 Battlefield Simulation (BATTLESIM)	2 - 14
2.4.3.1 BATTLESIM Architecture	2 - 15
2.4.3.2 Scenario File Organization	2 - 16
2.5 Summary	2 - 17
III. System Requirements and Design	3 - 1
3.1 Introduction	3 - 1
3.2 Definitions	3 - 1

3.3 Requirements ..... 3 - 2
3.3.1 Defining a Transitional Prototype ..... 3 - 2
3.3.1.1 Insufficiency of DIS PDU ..... 3 - 2
3.3.1.2 Defining a New Prototype ..... 3 - 3
3.3.2 Loading a source scenario ..... 3 - 4
3.3.3 Mapping to a Transitional Prototype ..... 3 - 5
3.3.4 Remapping to a Transitional Target Scenario Prototype ..... 3 - 6
3.3.5 Saving the Transitional Target Scenario Prototype ..... 3 - 8
3.4 Design ..... 3 - 8
3.4.1 Hardware Specification ..... 3 - 8
3.4.2 Software Specification ..... 3 - 9
3.4.2.1 General Organization ..... 3 - 9
3.4.2.2 User Interface ..... 3 - 9
3.4.2.3 Object Diagram ..... 3 - 10
3.4.3 Transitional Prototype Requirements ..... 3 - 11
3.4.3.1 Ideal Prototype ..... 3 - 11
3.4.3.2 Proposed Prototype ..... 3 - 11
3.4.4 Transitional Scenario Prototype Specification ..... 3 - 13
3.4.4.1 ModSAF ..... 3 - 13
3.4.4.2 EADSIM ..... 3 - 13
3.4.4.3 BATTLESIM ..... 3 - 15
3.4.4.4 Transitional Prototype ..... 3 - 16
3.4.5 Translation Tables ..... 3 - 18
3.5 Summary ..... 3 - 19

IV. System Implementation ..... 4 - 1

4.1 Introduction ..... 4 - 1
4.2 Loading a Scenario File ..... 4 - 1
4.3 Mapping a Scenario Prototype to a Transitional Prototype (TP) ..... 4 - 6
4.4 Remapping a TP to a Transitional Target Scenario Prototype ..... 4 - 10
4.5 Saving a TP as a Target Scenario ..... 4 - 16
4.6 User Interface ..... 4 - 17
4.6.1 Graphics User Interface ..... 4 - 17
4.6.2 Command Line Interface ..... 4 - 19
4.7 Summary ..... 4 - 19

V. Results ..... 5 - 1

5.1 Introduction ..... 5 - 1
5.2 The Scenario File Translator Test Cases ..... 5 - 1
5.2.1 Test 1 : ModSAF to EADSIM ..... 5 - 1
5.2.2 Test 2 : ModSAF to BATTLESIM ..... 5 - 1
5.2.3 Test 3 : ModSAF to ModSAF ..... 5 - 2
5.2.4 Test 4 : EADSIM to ModSAF ..... 5 - 2
5.2.5 Test 5 : EADSIM to BATTLESIM ..... 5 - 2
5.2.6 Test 6 : EADSIM to EADSIM ..... 5 - 3
5.2.7 Test 7 : BATTLESIM to ModSAF ..... 5 - 3
5.2.8 Test 8 : BATTLESIM to EADSIM ..... 5 - 3
5.2.9 Test 9 : BATTLESIM to BATTLESIM ..... 5 - 3
5.3 Observation ..... 5 - 4
5.4 Summary ..... 5 - 4

VI. Thesis Summary ..... 6 - 1

6.1 Introduction ..... 6 - 1
6.2 Recommendations for Future Work ..... 6 - 1
6.2.1 Transitional Prototype ..... 6 - 2
6.2.2 DIS PDU ..... 6 - 2
6.2.3 Machine Independent ..... 6 - 3
6.2.4 Compatibility ..... 6 - 4
6.2.5 Memory Size ..... 6 - 4
6.3 Conclusion ..... 6 - 4

Appendix A. Transitional Scenario File Prototype ..... A - 1
A.1 ModSAF ..... A - 1
A.2 EADSIM ..... A - 1
A.2.1 EADSIM Scenario File ..... A - 1
A.2.2 EADSIM Laydown File ..... A - 11
A.3 BATTLESIM ..... A - 21
A.4 Transitional File ..... A - 25

Appendix B. Enumeration Type for Task Type of Transitional Prototype ..... B - 1
B.1 Task Name ..... B - 1
B.2 Data Structures for Each Task Type ..... B - 2
B.3 Enumeration Type for Variables in Task ..... B - 12

Appendix C. Command Line Interface for SFT ..... C - 1

Bibliography ..... Bib - 1

Vita ..... Vita - 1

List of Figures

Figure		Page
1.1	Direct Simulation Conversion	1 - 4
1.2	Conversion from Simulation to Intermediate Prototype	1 - 5
2.1	ModSAF Simulated Battlefield	2 - 9
2.2	SAFstation and SAFsim Components of ModSAF	2 - 10
2.3	EADSIM Scenario Playback	2 - 12
2.4	EADSIM Model Processes and Applications	2 - 13
2.5	EADSIM Scenario Composition	2 - 14
2.6	BATTLESIM, TCHSIM, and SPECTRUM	2 - 16
3.1	Conversion using an Intermediate Format	3 - 5
3.2	Mapped and Unmapped Area into Intermediate Format	3 - 6
3.3	Common Area Between Intermediate Format and Target Scenario	3 - 7
3.4	Unrepresented Area In a Target Scenario	3 - 8
3.5	Translator General Diagram	3 - 9
3.6	Scenario Translator Object Diagram	3 - 11
3.7	Relationship between Simulations and Suggested Prototype	3 - 12
3.8	Transitional ModSAF Scenario Prototype	3 - 13
3.9	Transitional EADSIM 'Scenario File' Prototype	3 - 14
3.10	Transitional EADSIM 'Laydown File' Prototype	3 - 14
3.11	Transitional EADSIM Platform Prototype	3 - 15
3.12	Transitional BATTLESIM Prototype	3 - 16
3.13	Transitional Prototype mapping for ModSAF scenario	3 - 20
3.14	Transitional Prototype mapping for EADSIM scenario	3 - 21
3.15	Transitional Prototype mapping for BATTLESIM scenario	3 - 22
4.1	Algorithm for Reading a BATTLESIM Scenario file	4 - 4
4.2	Geocentric Cartesian Coordinates in DIS	4 - 7
4.3	Remapping to a Transitional ModSAF Prototype	4 - 13
4.4	Remapping to a Transitional EADSIM Prototype	4 - 14
4.5	Remapping to a Transitional BATTLESIM Prototype	4 - 15
4.6	The Main Window of the SFT	4 - 18
5.1	A Source Scenario of ModSAF	5 - 5

5.2 A Converted EADSIM Scenario from ModSAF ..... 5 - 5
5.3 A Converted ModSAF Scenario from ModSAF ..... 5 - 6
5.4 A Source Scenario of EADSIM ..... 5 - 6
5.5 A Converted ModSAF Scenario from EADSIM ..... 5 - 7
5.6 A Converted EADSIM Scenario from EADSIM ..... 5 - 7
5.7 A Converted ModSAF Scenario from BATTLESIM ..... 5 - 8
5.8 A Converted EADSIM Scenario from BATTLESIM ..... 5 - 8

List of Tables

Table		Page
2.1.	Entity State Protocol Data Unit	2 - 5
2.2.	Data Structure of ModSAF Scenario File	2 - 11
2.3.	BATTLESIM Input File	2 - 17
3.1.	Matching Comparison between Simulation Scenario and DIS	3 - 3
3.2.	Data Structure of Transitional File	3 - 17
3.3.	Percentages for Common Variables between Two Scenario	3 - 19
4.1	Required Memory Size for Loading a ModSAF Scenario File	4 - 3
4.2	Required Memory Size for Loading an EADSIM Scenario File	4 - 4
4.3	Required Memory Size for Loading a BATTLESIM Scenario File	4 - 5
4.4	Required Memory Size for Loading Transitional File	4 - 5
4.5	Icon Name in BATTLESIM vs. Entity Name in DIS	4 - 9
6.1	Proposed Future Work	6 - 3
A.1	ModSAF Transitional Prototype	A - 1
A.2	EADSIM 'Scenario File' Transitional Prototype	A - 1
A.3	EADSIM 'es_filename' data structure	A - 2
A.4	EADSIM 'es_header' data structure	A - 2
A.5	EADSIM 'es_laydownfile' data structure	A - 2
A.6	EADSIM 'es_elementmap' data structure	A - 2
A.7	EADSIM 'es_networkfile' data structure	A - 2
A.8	EADSIM 'es_zerothrumaxhost' data structure	A - 3
A.9	EADSIM 'es_unusedstring' data structure	A - 3
A.10	EADSIM 'es_debug' data structure	A - 3
A.11	EADSIM 'es_host' data structure	A - 4
A.12	EADSIM 'es_maxfilec3ioutput' data structure	A - 4
A.13	EADSIM 'es_truth' data structure	A - 4
A.14	EADSIM 'es_spds' data structure	A - 4
A.15	EADSIM 'es_log' data structure	A - 5
A.16	EADSIM 'es_stat' data structure	A - 5
A.17	EADSIM 'es_pstat' data structure	A - 5
A.18	EADSIM 'es_file' data structure	A - 6

A.19 EADSIM 'es_earthradadj' data structure ..... A - 6
A.20 EADSIM 'es_hzululocal' data structure ..... A - 6
A.21 EADSIM 'es_displaythruroute' data structure ..... A - 7
A.22 EADSIM 'es_formatthrustarting' data structure ..... A - 7
A.23 EADSIM 'es_process' data structure ..... A - 7
A.24 EADSIM 'es_write' data structure ..... A - 8
A.25 EADSIM 'es_c3ilog' data structure ..... A - 8
A.26 EADSIM 'es_image' data structure ..... A - 9
A.27 EADSIM 'es_transradfilepath' data structure ..... A - 9
A.28 EADSIM 'es_defaultsentime' data structure ..... A - 9
A.29 EADSIM 'es_defaultjamtime' data structure ..... A - 9
A.30 EADSIM 'es_totalruns' data structure ..... A - 10
A.31 EADSIM 'es_time' data structure ..... A - 10
A.32 EADSIM 'es_external' data structure ..... A - 10
A.33 EADSIM 'es_tb data structure ..... A - 11
A.34 EADSIM 'es_lay' data structure ..... A - 11
A.35 EADSIM 'es_layplatform' data structure ..... A - 12
A.36 EADSIM 'es_laysensor' data structure ..... A - 13
A.37 EADSIM 'es_laysensorheader' data structure ..... A - 13
A.38 EADSIM 'es_laycomdev' data structure ..... A - 14
A.39 EADSIM 'es_layjammer' data structure ..... A - 14
A.40 EADSIM 'es_layjammerheader' data structure ..... A - 14
A.41 EADSIM 'es_layasset' data structure ..... A - 15
A.42 EADSIM 'es_laytarget' data structure ..... A - 16
A.43 EADSIM 'es_layiftuplatform' data structure ..... A - 16
A.44 EADSIM 'es_layplatformsatellite' data structure ..... A - 16
A.45 EADSIM 'es_layuseroutedata' data structure ..... A - 17
A.46 EADSIM 'es_laynotuseroutedata' data structure ..... A - 17
A.47 EADSIM 'es_layptl' data structure ..... A - 17
A.48 EADSIM 'es_layacfrc' data structure ..... A - 17
A.49 EADSIM 'es_laywaypoint' data structure ..... A - 18
A.50 EADSIM 'es_layart' data structure ..... A - 18

A.51 EADSIM 'es_layplatformaircraft' data structure ..... A - 18
A.52 EADSIM 'es_layplatformairbase' data structure ..... A - 19
A.53 EADSIM 'es_laymilid' data structure ..... A - 19
A.54 EADSIM 'es_laytime' data structure ..... A - 19
A.55 EADSIM 'es_laypointing' data structure ..... A - 20
A.56 EADSIM 'es_laynurdwaypoint' data structure ..... A - 20
A.57 EADSIM 'es_layaircraftdata' data structure ..... A - 20
A.58 BATTLESIM 'bs_scenario' data structure ..... A - 21
A.59 BATTLESIM 'BS_TerrainMinCoord' data structure ..... A - 21
A.60 BATTLESIM 'BS_TerrainMaxCoord' data structure ..... A - 21
A.61 BATTLESIM 'bs_sector' data structure ..... A - 21
A.62 BATTLESIM 'bs_icon' data structure ..... A - 22
A.63 BATTLESIM 'BS_Header' data structure ..... A - 22
A.64 BATTLESIM 'bs_object' data structure ..... A - 22
A.65 BATTLESIM 'BS_ObjectLocation' data structure ..... A - 22
A.66 BATTLESIM 'BS_ObjectVelocity' data structure ..... A - 23
A.67 BATTLESIM 'BS_ObjectOrientation' data structure ..... A - 23
A.68 BATTLESIM 'BS_DescObj' data structure ..... A - 23
A.69 BATTLESIM 'bs_routepoint' data structure ..... A - 24
A.70 BATTLESIM 'bs_sensor' data structure ..... A - 24
A.71 BATTLESIM 'bs_armament' data structure ..... A - 24
A.72 BATTLESIM 'bs_target' data structure ..... A - 24
A.73 BATTLESIM 'bs_defensive' data structure ..... A - 25
A.74 Transitional File 'tf_transfile' data structure ..... A - 25
A.75 Transitional File 'tf_database' data structure ..... A - 25
A.76 Transitional File 'tf_point' data structure ..... A - 26
A.77 Transitional File 'tf_location' data structure ..... A - 26
A.78 Transitional File 'tf_entityheader' data structure ..... A - 26
A.79 Transitional File 'tf_entitytype' data structure ..... A - 26
A.80 Transitional File 'tf_entityvelocity' data structure ..... A - 26
A.81 Transitional File 'tf_entityorientation' data structure ..... A - 27
A.82 Transitional File 'tf_munitiontype' data structure ..... A - 27

A.83 Transitional File 'tf_sensorid' data structure ..... A - 27
A.84 Transitional File 'tf_jammerid' data structure ..... A - 27
A.85 Transitional File 'tf_tasktype' data structure ..... A - 27
B.1 Enumeration Type of Task Name ..... B - 1
B.2 Task 'Move' data structure ..... B - 2
B.3 Task 'Follow Simulator' data structure ..... B - 3
B.4 Task 'Hasty Occupy Position' data structure ..... B - 3
B.5 Task 'Assault' data structure ..... B - 4
B.6 Task 'Traveling Overwatch' data structure ..... B - 4
B.7 Task 'Overwatch Movement' data structure ..... B - 4
B.8 Task 'Withdraw' data structure ..... B - 5
B.9 Task 'Breach' data structure ..... B - 5
B.10 Task 'Concealment' data structure ..... B - 6
B.11 Task 'Delay' data structure ..... B - 6
B.12 Task 'Repair' data structure ..... B - 6
B.13 Task 'Service Station' data structure ..... B - 6
B.14 Task 'Cross Leveling' data structure ..... B - 7
B.15 Task 'Change Formation' data structure ..... B - 7
B.16 Task 'Rendezvous' data structure ..... B - 7
B.17 Task 'FWA Sweep' data structure ..... B - 7
B.18 Task 'FWA CAP' data structure ..... B - 8
B.19 Task 'FWA CAS Mission' data structure ..... B - 8
B.20 Task 'FWA Ingress' data structure ..... B - 9
B.21 Task 'FWA Attack Ground Target' data structure ..... B - 9
B.22 Task 'weaponsenabled' data structure ..... B - 9
B.23 Task 'FWA Return to Base' data structure ..... B - 10
B.24 Task 'FWA Interdiction' data structure ..... B - 10
B.25 Task 'RWA Fly Route' data structure ..... B - 10
B.26 Task 'RWA Hover' data structure ..... B - 11
B.27 Task 'RWA Orbit' data structure ..... B - 11
B.28 Task 'RWA Attack' data structure ..... B - 11
B.29 Task 'RWA Hasty Occupy Position' data structure ..... B - 11

List of Abbreviations

ADS	Advanced Distributed Simulation
AFIT	Air Force Institute of Technology
ARPA	Advanced Research Projects Agency
BATTLESIM	Battlefield Simulation
C2	Command and Control
C3I	Command ,Control, Communications, and Intelligence
C4I	Command ,Control, Communications, Computers, and Intelligence
CAS	Close Air Support
CCTT	Close Combat Tactical Trainer
CGF	Computer Generated Forces
CLI	Command Line Interface
DARPA	Defense Advanced Research Projects Agency (now ARPA)
DES	Discrete Event Simulation
DIS	Distributed Interactive Simulation
EADSIM	Extended Air Defense Simulation
FWA	Fixed Wing Aircraft
GUI	Graphics User Interface
ID	Identity
IEEE	Institute of Electrical and Electonics Engineering
LP	Logical Process
METT-T	Mission, Enemy, Troops, Terrain and Time
ModSAF	Modular Semi-Autonomous Forces
NEQ	Next Event Queue
PASE	Parallel Ada Simulation Environment
PDES	Parallel Discrete Event Simulation
PDU	Protocol Data Unit
PO	Persistent Object
QUEUESIM	Queuing Model Simulation
RWA	Rotated Wing Aircraft
SAF	Semi-Autonomous Forces

SFT	Scenario File Translator
SGI	Silicon Graphics Incorporation
SIMNET	Simulation Network
SPECTRUM	Simulation Protocol Evaluation Testbed using Reusable Models
STOW	Synthetic Theater of War
STRICOM	U.S. Army Simulation, Training and Instrumentation Comand
TCHSIM	Thomas C Hartrum Simulation
TP	Transitional Prototype
USA	United States of America
USSR	Union of Soviet Socialist Republics
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VSIM	VHDL Simulation
VV&A	Verification, Validation & Accreditation

Abstract

There exists a variety of simulation generation and analysis products which have differing purposes and functions designed to simulate a real military battlefield. Due to the particular purpose of each simulation, it is impractical to use one scenario of a simulation directly with another simulation without a translator since there is no standard scenario format. In the current environment, interoperability between simulations is becoming more important in large scale simulations and distributed exercises. The Scenario File Translator (SFT) provides an easy and accurate way to create a scenario from a heterogeneous simulation. The SFT can load and save the three research simulations: ModSAF, EADSIM, and BATTLESIM. It also defines the general transitional prototype (TP) which is the information most commonly used to create a mock battlefield computer simulation. System functionality is accessible through a graphical user interface (GUI). The four top-level steps to translate a scenario are loading a source scenario file, mapping to the TP, remapping the TP to a transitional target prototype, and saving the prototype as a target scenario file which a heterogeneous simulation use. Although this system was designed for three simulations, it can be applied to any other simulations by creating only two additional functions: one which maps the new scenario to the TP and another which remaps the TP into the transitional scenario protocol.

THE DEVELOPMENT OF A SCENARIO TRANSLATOR FOR DISTRIBUTED SIMULATIONS

I. Introduction

1.1 Background

Most people want maximum results with minimum investment in cost and time when doing a work. Attempts to attain such results will continue and computer simulation is a good way to accomplish it. Computer simulation is very useful when the real system does not exist or it is expensive, time consuming, dangerous, or impossible to build [Neel87]. The effectiveness of computer simulation in the military is especially apparent, since real combat conditions are often costly to recreate and often dangerous. When we use computer simulation, we do not have to fly many aircraft nor send hundreds of tanks to attack a target, furthermore we do not have to maneuver thousands of soldiers to other places for training. Computer simulation also allows testing of many scenarios in a short time so that we can measure which scenario is the best model at a certain place.

To simulate a real battlefield, some preconditions should be solved, such as an accurate terrain database, a precise description of strategy, and a stable network. High speed computer networks and distributed simulation applications make possible computer simulations for the military. Even though the opposing forces may be located far away, the high speed communication network provides real time simulation as if each side is close at hand. Thus the exerciser can develop fighting skills through mutual exercises. However, the important thing here is that a standard communication protocol is necessary to implement the

distributed communication. For example, assume exercise *A* uses protocol X and exercise *B* uses protocol Y: how can exercise *A* know exercise *B* fires a missile, or is fired upon? Developing a well-defined standard protocol is as important as good simulation software. As a result, the Defense Advanced Research Projects Agency (DARPA, now called ARPA) made a standard communication protocol, which was named the Simulation Network (SIMNET). Between 1983 and 1989, ARPA successfully demonstrated the core technology for networking large numbers of manned, homogeneous simulators using SIMNET. Distributed Interactive Simulation (DIS) standards are being developed to provide industry wide standards that enable linking of heterogeneous systems. Since 1992, the DIS standards have been demonstrated numerous times. The demonstrated scenarios displayed maritime, air-to-air, ground-to-air, air-to-ground, and land operations. These demonstrations proved the viability of linking simulations of different types, based on different technologies, and built by different organizations [DIS 94].

The past few years have seen remarkable advances in DIS. The DIS environment allows military units from around the country to participate in mock training exercises without a large commitment of resources [Adam93]. As mentioned earlier, there now exists a variety of scenario generation and analysis products such as Integrated Eagle, Janus, CCTT (Close Combat Tactical Trainer), Modular Semi-Autonomous Forces (ModSAF), and Synthetic Theater of War (STOW). Each simulation has its differing purposes and functions. So after testing a scenario with different simulations, we can build better combat models and get more precise verification.

However, a basic problem still exists. Due to the particular purpose of each simulation, we cannot use one scenario of a simulation directly with another simulation without a translator since there is no standard scenario format. To test a new scenario, it is

impractical to manually recreate it to make it fit another simulation. In the current environment, interoperability between simulations is becoming more important in large scale simulations and distributed exercises. Since current scenarios can contain thousands of entities, scenario setup and initialization can be quite laborious.

1.2 Problem Statement

There exist several ways to apply a battle scenario of one simulation to other simulation. The most straight forward way is to recreate the scenario file in a format suitable for the other simulation, but this is time consuming work and an ineffective. If we want to apply a scenario to five different simulations, we need to make five versions of the same scenario for each simulation. If we want to test 10 scenarios for these same five simulations, then 50 completely different scenarios are needed. There could be ten thousand entities in a scenario, so it is not easy work to make a complete scenario. Making manual conversion of such scenarios is inefficient. Another solution for these problems is a translation tool to automatically convert scenarios from one simulation format to another. I propose to produce a software tool that aids in the translation of scenario file formats so that scenarios developed by different simulations can be run by each other. A well-defined translator will save time and money, and it will provide the best way to test a scenario with different simulations.

1.3 Assumption

Most simulations have a specific purpose architecture. As the goal of each simulation is different from others, the way to read or write a scenario file may also be different. For example, the amount of munitions for a unit may be represented in one simulation, but not another. So, translating one scenario format into a different format is not guaranteed to

preserve all information. That means some data which is not present in a target simulation can be lost while converting to another scenario. The important thing here is the conversion rate which shows how many entities are translated to the other simulation.

1.4 Approach

This section discusses some key decisions I used to accomplish this work. First of all, I had to choose a conversion method to translate one scenario to another. Based on this decision, I designed an intermediate format covering one scenario of each simulation. Then in the remainder of the section, I delineated detailed processes.

1.4.1 Determining Conversion Method A problem with writing a generalized scenario translator is that the data structures used to initialize scenarios in different tools are very specialized. One way to solve this problem is to develop a tool which can map directly between any two simulations. This increases the conversion rate. This concept is depicted in Figure 1.1.

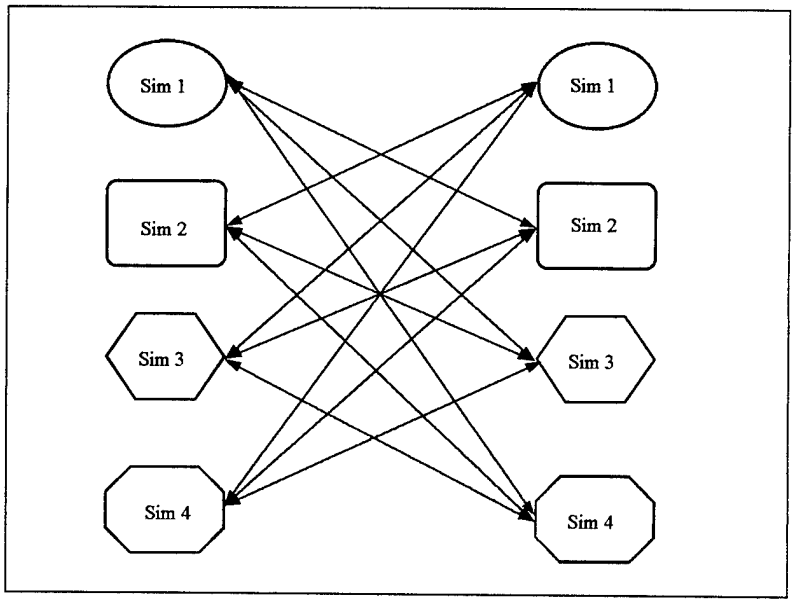

Figure 1.1 Direct Simulation Conversion

This concept is efficient when the number of simulations is small. However, whenever a new simulation is developed, it is necessary to write a new conversion routine. This routine translates the new scenario of the new simulation to each existing simulation: I also needed to create a routine that converts a scenario of an existing simulation to the new simulation. Assume there are currently *N* simulations. When a new simulation is developed, *N* routines are needed to map a scenario of the new simulator into the existing simulation, and another *N* routines are necessary to map the scenario of the existing simulations to the one of the new simulations. Thus a total of *2*N* routines are needed.

Another way to do this is to do remap to a target simulation after mapping into an intermediate prototype. This is more efficient because it is not dependent on the number of existing simulations. This requires two routines: a routine that maps to the intermediate prototype, and a routine that remaps to the target scenario of the simulation. Thus only two routines are always needed, compared to the *2*N* routines required using the previous method. Figure 1.2 shows this concept.

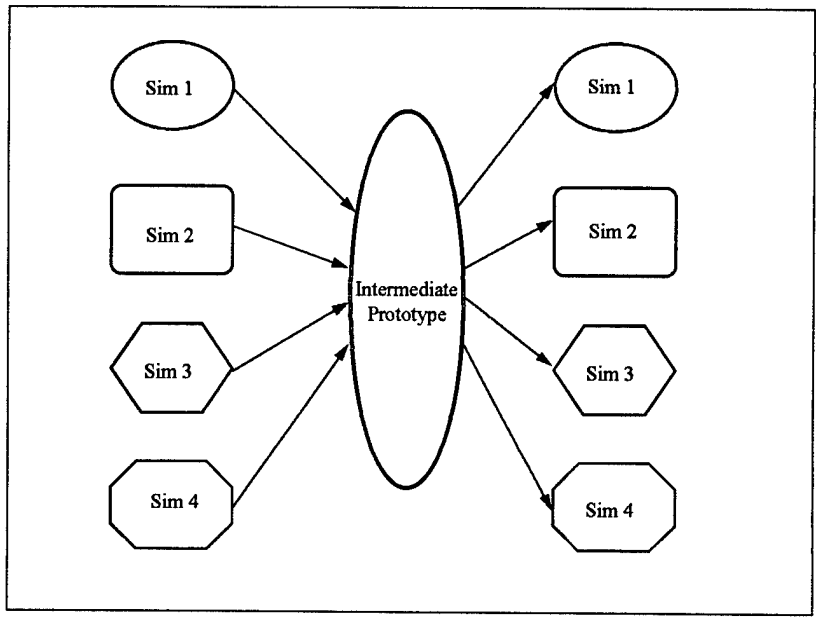

Figure 1.2 Conversion from Simulation to Intermediate Prototype

I propose that this translation be done in two steps. The first step is mapping a source scenario to an intermediate transitional prototype. The second step is saving the intermediate prototype to the target scenario format used to run target simulations.

1.4.2 Mapping to Transitional Prototype In order to develop a reasonable mapping, I first began with a comprehensive survey of the three target research simulations: ModSAF, EADSIM, and BATTLESIM. Furthermore, I had organized the survey as the following set of tasks.

- Review of Semi-Automated Forces (SAF)
  First, I reviewed the history of SAF technology and current research efforts in the field. That review gave me a basic knowledge of the simulations.

- Survey of current simulations
  Next, I investigated each simulation's basic architecture and scenario's formation. It was beneficial for me to understand what things were common and what points were unique.

- Finding the Transitional Format
  Finding the transitional format was the most important part of this research. By choosing a well-designed transitional format, I could make a more generic program. The way to find an intermediate transitional prototype that suits this concept is either to select a protocol among these currently developed or to define a new protocol fitting the above concept. A more detailed discussion of this topic appears in Section 3.3.1.

- Build the Comprehensive Table
  Following these steps, I composed a table suitable to compare each simulation. This table gave me the insight into what data are in common and which data structure is more efficient to translate a scenario.

1.4.3 Remapping to Target Scenario

The remapping step to a target scenario using the comparison table was the index of performance for this research effort. As I mentioned in Section 1.3, it was unrealistic to expect 100 % conversion; however, a "close" scenario was likely, and it was the goal of this research to determine how "close" a conversion the transitional prototype could support. This goal was implemented through the next two steps.

- Find the common data between target scenario and the transitional prototype.
- Save as a new scenario file which can be read by a target simulation.

1.5 Thesis Overview

This thesis is divided into six chapters. Chapter Two presents the background material on Advance Distributed Simulation, and DIS and Computer Generated Forces (CGF) or SAFs. Chapter Three focuses on the requirements definition and analysis process and its translation into a system design. Chapter Four discusses implementation of the system and describes how it works. Chapter Five summarizes the results and illustrates several test cases. Finally, Chapter Six provides recommendations for future work and the conclusion.

II. Background and Literature Review

2.1 Introduction

This chapter summarizes the work in the two major research areas that is relevant to my thesis effort. Since the primary purpose of this thesis was to create a scenario translation tool, I completed a survey of the enabling and supporting technology in that field. The first section provides a historical background how DIS was evolved, and the next section lays the foundation for understanding the important concepts in military simulation. Finally, the last section discusses three of the simulations recently developed – ModSAF, EADSIM and BATTLESIM – and focuses on how each simulation works and what are the differences among them.

2.2 Historical Background

To date, various research efforts have focused on developing war game simulations which have a particular ability for the given research. New simulations with new abilities are created one after another, and existing simulations are upgraded to be more accurate and realistic. The one point that all simulations have in common is standardization through the network for sharing data. This standardization has been accomplished using DIS.

DIS had been started in 1989 when ARPA initiated a program to enhance the Simulation Network (SIMNET) program. SIMNET was developed for building a cross-country network of interactive combat simulators [Pope91]. Its demonstration of approximately 250 simulators successfully showed the possibilities of distributed simulation. SIMNET technology is still used today to train U.S. Army tank teams around the country.

DIS is a set of protocols that carry messages about entities and events through a network. When a network provides reasonably low latency (100 to 300 milliseconds) and low

latency variance, the characteristics of the network present few problems to building a virtual world that is separated by thousands of miles in the real world [DIS94]. One important aspect of DIS development is the effort to recognize DIS as an international standard for distributed simulation. Thus, the DIS standard was submitted and approved by the Institute of Electrical and Electronic Engineers (IEEE) as IEEE standard 1278. Following IEEE approval, the standard was submitted to the appropriate international agencies for international standard approval. This is useful, since it promotes cooperation among the U.S. and its allies in the field of simulation [McDo91].

2.3 Military Simulation Technology

2.3.1 Distributed Interactive Simulation (DIS)

2.3.1.1 Concept DIS is an interconnected, time-coherent simulation system which creates a distributed, interactive environment using the IEEE 1278 protocol [Siko95]. DIS simulators exchange information in formatted messages called Protocol Data Units (PDUs). These PDUs provide data for the management and control of a DIS exercise and provide data concerning simulated entity states and the types of entity interactions that take place in a DIS exercise. It is possible for geographically separated simulators to interact with each other via network communications by the DIS standard [IST94]. DIS is designed for linking the interactive, free play activities of people in operational exercises to represent a time and space coherent synthetic world environment. This environment is created through real-time exchange of data units between distributed, computationally autonomous simulation applications in the form of simulation, simulators, and instrumented equipment interconnected through standard computer communicative services. The meaning of "distributed" is that no single computer controls the simulations. Rather, each local computer is responsible for sending local copies of common terrain and models

(e.g., tanks, fighters, and naval vessels), and remote entities based on incoming PDU messages. Local applications are required to send entity updated entity states when dead-reckoning thresholds are exceeded or five seconds have elapsed [Gard93, Shea92]. Entities' positions and orientations are predictable by using the Dead Reckoning algorithm so that the number of transmitted PDUs can be kept as low as possible. This increases the scale of the exercise and reduces network communications traffic from a single simulator [Gard93]. Dead Reckoning is an important algorithm as one of the basic concepts underlying the DIS architecture. It also diminishes the computational processing associated with receipt of each new PDU because fewer PDUs are received [Harv91].

2.3.1.2 DIS Objectives Principles of the emerging DIS standards and their applications are introduced in this section. Basic architectural concepts include [IST94]:

- No central computer controls the entire simulation exercise,
- Autonomous simulation applications are responsible for maintaining the state of one or more simulation entities,
- A standard protocol is used for communicating "ground truth" data,
- Changes in the state of an entity are communicated by simulation applications,
- Perception of events or other entities is determined by the receiving application, and
- Dead reckoning algorithms are used to reduce communications processing.

A definition of each of these concepts is provided in the proposed IEEE draft in reference [IST93].

2.3.1.3 DIS PDUs DIS currently defines 27 different PDUs. A complete description of each of these PDUs can be found in reference [IST94] and can be grouped into the following general categories:

- Entity Information/Interaction,

- Warfare,
- Logistics,
- Simulation Management,
- Distributed Emission Regeneration, and
- Radio Communications.

Only 12 of these 27 PDUs have a fixed length. Each of the remaining PDUs has a variable length dependent upon many factors including numbers of articulated parts, numbers and types of emitters, length of audio transmissions, and amount of data.

Table 2.1 [IST94] depicts the entity state PDU as an example of the content and format of the DIS PDUs.

2.3.2 Advanced Distributed Simulation (ADS) Computer simulation has been used by military analysts in various aspects since the 1950s, especially in mock battle areas. At the beginning, it had very limited scope due to the vastness of the battle and the complexity of the simulated war, which also made it difficult to use and to satisfy the users. The development of larger mainframe computers in the 1960s resulted in enlarged storage capacity and greater processing speed allowing better digitized terrain and larger scenarios than before. By the 1980s, the capabilities of computer-based simulations had really expanded with the advent of microprocessor chips, high speed data communication links, larger mass storage devices, and flexible, detailed displays. all at low cost. Today, the simulated battlefield can depict more accurately real-time war games because of continually improving computer capabilities. As the SIMNET and DIS demonstration showed, it is possible to have different simulated battlefields at dispersed geographic locations working together. The new capabilities, such as high-speed communication networks, common network interface/translation devices, and emerging standard data protocols, have allowed a time-coherent, interactive synthetic

Table 2.1. Entity State Protocol Data Unit

Word	Field	Byte 0	Byte 1	Byte 2	Byte 3
0	PDU Header	Version	Exer ID	PDU Type	Family
1		Time Stamp			
2		Length		Pad	
3	Entity/Force ID	Site		Application	
4		Entity		Force ID	# Artic Parts
5	Type	Kind	Domain	Country	
6		Category	Subcategory	Specific	Extra
7	Alt Type	Kind	Domain	Country	
8		Category	Subcategory	Specific	Extra
9-10	Linear Velocity	X Component			
11-12		Y Component			
13-14		Z Component			
15	Location	X Component			
16		Y Component			
17		Z Component			
18	Orientation	Psi			
19		Theta			
20		Phi			
21	Appearance	Appearance			
22	Dead Reckon Parms	Algorithm	Unused		
23-24		Unused			
25		Linear Accel X Component			
26		Linear Accel Y Component			
27		Linear Accel Z Component			
28		Angular Velocity X Component			
29		Angular Velocity Y Component			
30		Angular Velocity Z Component			
31	Entity Marking	Char Set	Marking		
32-33		(Marking continued)			
34	Capabilities	Boolean fields			
35	Artic Parms	Designator	Change	ID	
36		Parameter Type			
37-38		Parameter Value			

environment where computer simulation, real equipment and systems, and humans work together in synthetic battlefields for training, system development, testing, and force assessment. This synthetic environment through geographically distributed and using potentially dissimilar simulation hardware and software is a technology area named Advanced Distributed Simulation (ADS) [Siko95].

There are many goals in ADS. The main goal is to create a synthetic world distributed over a global network, which is realistically populated with: high resolution, dynamic terrain; tactically significant environmental effects; individual combatants and weapon platforms [Garr95]. Three simulation types including these main components are discussed below. The first one is '*live simulation*' in which real people operate real systems in actual operational conditions. For example, an operational test of the F-22 in an electronic combat environment in Nevada is an example of a live simulation. The second one is '*virtual simulation*' in which real people operate simulated systems. An example is a testbed such as Advance Research Project Agency's (ARPA) Simulation Network (SIMNET). However, the most important and capable examples of this type are represented by Distributed Interactive Simulation (DIS) systems. Finally, those simulations in which simulated people operate simulated systems are called '*constructive simulations*'. Computer generated forces such as Modular Semi-Automated Forces (ModSAF) are good examples of this type of simulation.

One of the important functions of ADS is to 'talk to and understand' what other connected simulations are doing and what their simulated elements are doing. If the simulations which compose a specific ADS configuration are all of the same type, scope, and data structure, the communications interface between them is relatively straight-forward. However, most simulations in an ADS have different scopes and data structures. So a common language (or protocol) is necessary to enable communication with all connected simulations. The most widely recognized and most comprehensive effort to define an ADS standard simulation protocol is the DIS protocol. Development of this protocol occurs as a cooperative effort by government and industry through Forms called the Distributed Interactive Simulation Workshops [Siko95].

2.3.3 Computer Generated Forces (CGF) When the Army Modeling and Simulation Master Plan was published in 1994, it named 20 areas for consideration in the Army modeling and simulation community. The Master Plan defined the implementing software standards across the modeling community and encouraged standards for development through the establishment of "teaming arrangements" and "consensus building" within the Army modeling community. CGF is the one of those areas. CGF Systems originated in the SIMNET environment in the mid 1980s. Their initial use was to provide a set of threat vehicles and live friendly forces to train personnel in the SIMNET simulators. Since 1990, CGF systems have been used in other ways, providing both friendly and threat forces in virtual experimentation environment. These CGF-based virtual battles have been interfaced with sensor simulators (J-STARS) and live personnel in the command and control (C2) network making decisions on interdiction of deep targets. The following list is a minimal set of objectives for any CGF system as a "standard" [Pick95]:

- The CGF system must be useful to all three applications: training, advanced technology demonstration, and analysis. This objective implies specific requirements such as:
  - Ability for man-in-loop simulators to interface at any echelon.
  - Ability to interface with live systems at any echelon
  - Ability to run real-time and, for analytic applications, faster than real-time.
  - Ability to interface with constructive models in the constructive+virtual environment.
- The CGF system must be DIS compatible and also have the ability to operate across both local and wide area nets.
- The CGF system must represent the battle from Corps to individual vehicle.
- The CGF system must interface with other Service models in a Joint exercise.
- The number of operators in the CGF system must be minimal.
- The structures and data bases simulating the physical and cognitive/tactical behaviors of the vehicles, personnel and units must be modular and easily isolated for Verification, Validation & Accreditation (VV&A).

For the architectural development of any CGF system, the design of certain components are needed, which provide the simulation with the ability to represent the physical

environment in which the battle will take place, to represent the combatants themselves and to represent a C2 structure for organizing the individual combatants into units and as a single fighting force. The CGF system also demands the development of services supporting a distributed simulation. The goals for depicting a CGF simulated environment are to represent the individual vehicular commander to perform METT-T (Mission, Enemy, Troops, Terrain and Time) activities, to represent the effects of weather, dynamic changes in terrain, and battlefield haze and smoke.

2.4 Distributed Simulations

This section gives a brief synopsis of some SAF simulations that have been recently developed such as ModSAF, EADSIM and BATTLESIM.

2.4.1 Modular Semi-Automated Forces (ModSAF)

ModSAF is the successor to the SIMNET and ODIN Semi-Automated Forces systems. ModSAF 1.0 was released in December 1993 following the development of the ModSAF architecture in the spring of 1992 under sponsorship by ARPA/ASTRO and STRICOM [Cour95]. ModSAF is a combat simulation that supports DIS. It lets us create and control combat entities on a simulated battlefield. These entities replicate the outward behavior of their component vehicle and weapon systems to a level of realism sufficient for training and combat development [ADST95]. Figure 2.1 shows an actual ModSAF simulated battlefield.

2.4.1.1 ModSAF Architecture There are three components in the ModSAF architecture: (1) SAFstation, (2) SAFsim, and (3) Logger. These components are typically run on separate computers over a network; however, the SAFsim and SAFstation can run on

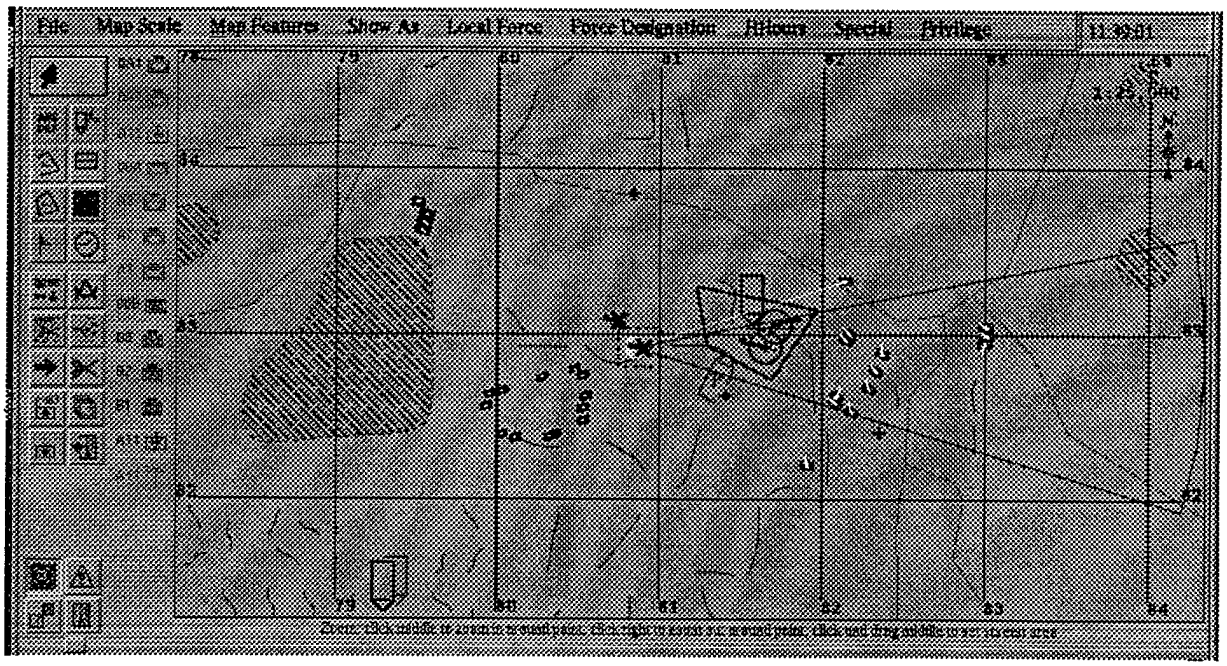

Figure 2.1. ModSAF simulated battlefield

the same computer. The components communicate physical battlefield state and events among themselves through the simulation (DIS) protocol and command, control, and system information through the Persistent Object (PO) protocol [ADST95].

Figure 2.2 shows the SAFstation and SAFsim components communicating over a network via PO and simulation packets that are marked with the same exercise ID [ADST95]. SAFsim and SAFstation communicate command, control, and system information via PO packets for bundling graphic, unit, model parameter, task, task frame, and exercise data. When we create objects on the SAFstation, PO packets which are representing the state of each object are projected onto the network so that the objects they represent can be simulated by the SAFsim. SAFsim and SAFstation communicate physical battlefield state and events between themselves via simulation packets for entity state, impact, collision, fire, initialization, radar, and weather data [ADST95].

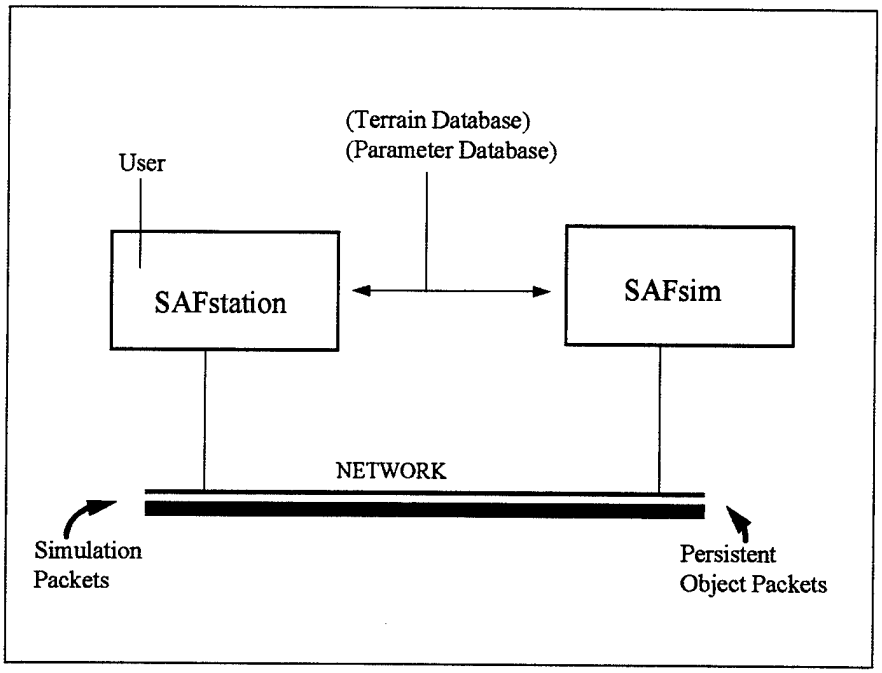

Figure 2.2 SAFstation and SAFsim components of ModSAF

The Parameter Database contains the modeling parameters used in the construction of the ModSAF system. This set of parameter files allows modification of the ModSAF system without further computer programming. Both the SAFstation and the SAFsim components have access to a Terrain Database. The SAFsim simulates objects known as ModSAF 'entities' (such as planes and tanks) which can behave autonomously. When a SAFsim simulates a unit, the SAFsim not only creates the entities in the unit, it also builds a structure corresponding to the unit hierarchy.

2.4.1.2 Scenario File organization ModSAF uses only one file to present a battle scenario that might have hundreds or thousands of entities. ModSAF 2.1 defines 21 classes to describe these entities. A scenario file consists of two parts: a header part and an entity part. The

header part includes some global data such as version number, number of objects and number of entries. The rest of a scenario file is all about these entries. ModSAF reads a scenario file sequentially according to this number of entries. An entity part also consists of two parts: the file entry part and the class part. The important data of the file entry part is the size of class which follows the entry part. Table 2.2 shows this data structure of a scenario file. As each class from the scenario file is read, one large battle scenario is created.

Table 2.2 Data Structure of ModSAF scenario file

Name		Data of Interest
Header Part		File Format Protocol Version Database Version Number of Objects Number of Entities
0	File Entry	Serial Number Variant Size
	Class	
1	File Entry	
	Class	
continue by number of objects in header part		

2.4.2 Extended Air Defense Simulation (EADSIM)

EADSIM is a simulation of air and missile warfare used for scenarios ranging from few-on-few to many-on-many. Each platform (such as a fighter aircraft) is individually modeled and the interaction among the platforms is also individually modeled. EADSIM also models the Command and Control (C2) decision processes and the communications among the platforms on a message-by-message basis. Intelligence gathering is explicitly modeled as is the

intelligence information used in both offensive and defensive operations [TBE95]. The general areas modeled are: air defense, offensive air operations, attack operations, multi-stage ballistic missiles, air breathers, sensors, jammers, satellites, early warning, generic noncombatants, communications, electronic warfare, terrain, weaponry, and areas of interest. Figure 2.3 depicts an EADSIM Scenario playback.

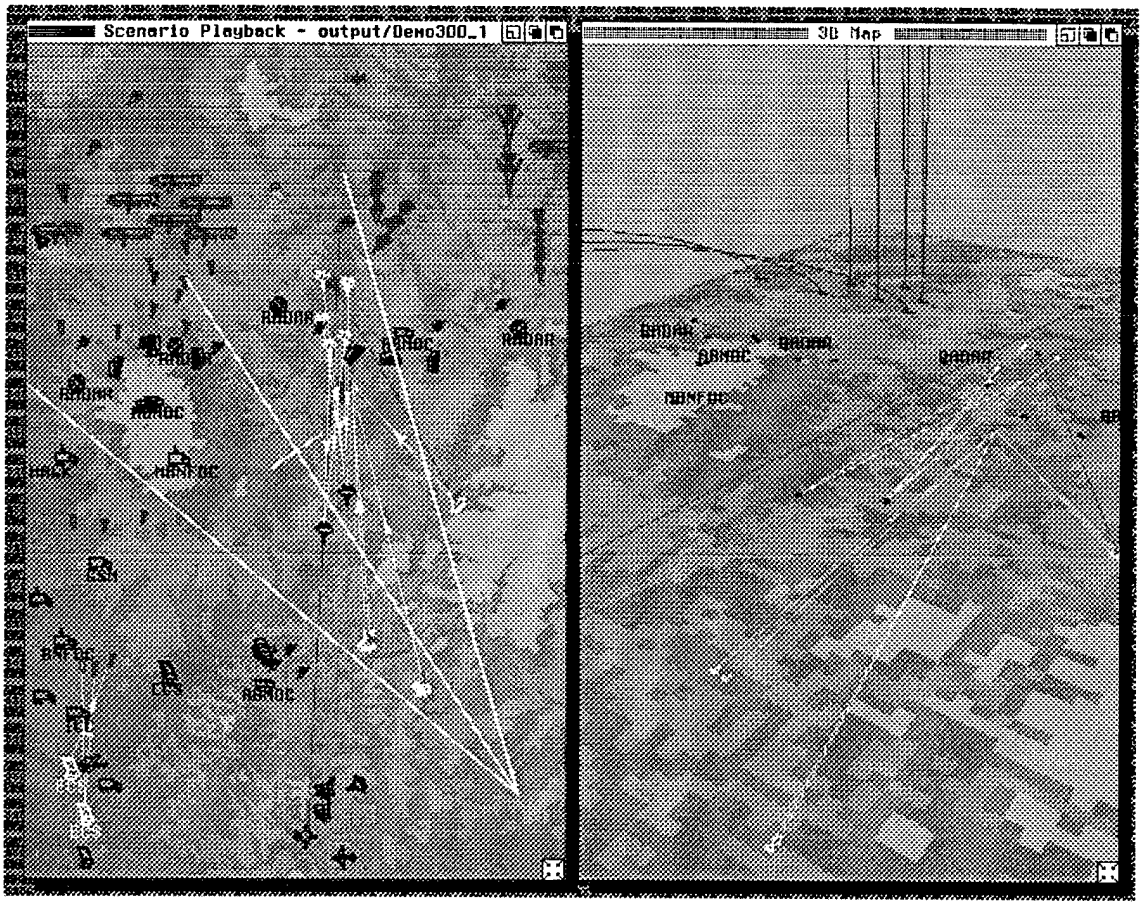

Figure 2.3 EADSIM Scenario Playback

2.4.2.1 EADSIM Architecture The EADSIM model consists of numerous processes and process applications which perform three basic functions: simulation setup, execution of a scenario, and post-processing and analysis. The simulation setup and post-processing and

analysis tools are run on a graphic window manager which provides the primary user interface. The execution of a scenario is performed by a set of run-time models running in a multi-process configuration. Figure 2.4 shows the processes and applications making up each of these areas [TBE95].

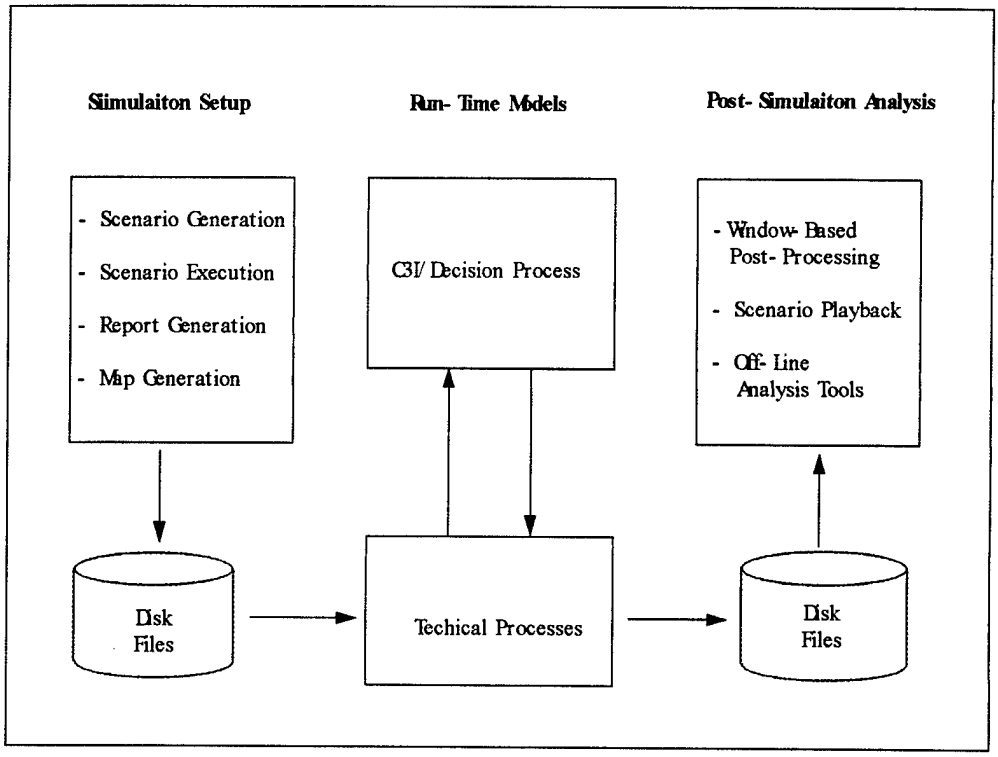

Figure 2.4 EADSIM Model Processes and Applications

2.4.2.2 Scenario File Organization A scenario file is made by EADSIM's Scenario Generation. The data of the scenario is organized in a hierarchical structure on which each level of the hierarchy is built on lower levels. The lowest level in the hierarchy is the elements. Combinations of these individual component elements are then used to organize System Elements that are deployed to form Platforms. Groupings of Platforms are built into Laydowns, where the Platforms in the Laydowns are interconnected with Networks which also

use the Protocol elements. This Platform level corresponds to entities which most people know. Areas of Interest (AOIs) can be created and associated with both Platforms and Networks. The Map specification forms the geographic basis for the scenario. The scenario is then a further combination of all of the lower level data [TBE95]. Figure 2.5 [TBE95] is a diagram representing the data organization.

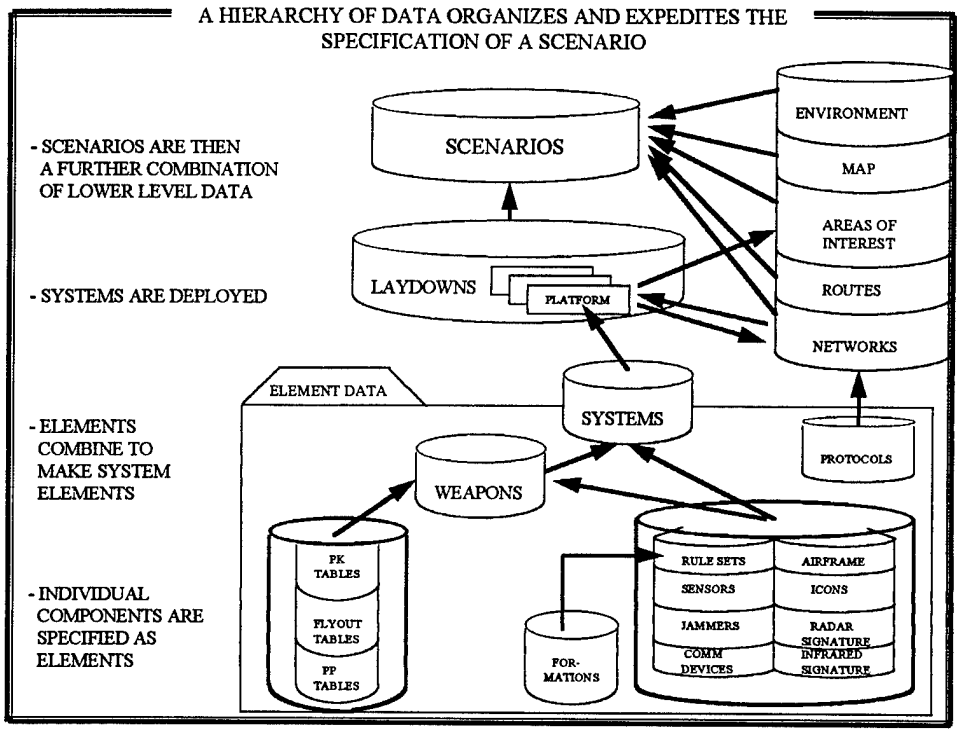

Figure 2.5 EADSIM Scenario Composition

2.4.3 Battlefield Simulation (BATTLESIM)

BATTLESIM is one of the Air Force Institute of Technology (AFIT) Parallel Discrete Event Simulation (PDES) research efforts. The primary discrete event simulation (DES) environments available for use at AFIT are Battlefield Simulation (BATTLESIM), the Parallel Ada Simulation Environment (PASE), Very High Speed Integrated Circuit (VHSIC)

Hardware Description Language (VHDL) Simulation (VSIM), and Queuing Model Simulation (QUEUESIM). These environments have evolved over the course of the AFIT parallel simulation research program to employ parallelism using the Chandy-Misra conservative synchronization approach as implemented in the SPECTRUM (Simulation Protocol Evaluation Testbed using Reusable Models) host manager [Hill94]. DES is an event-driven simulation approach in which the simulation changes state upon event execution [Hill94]. PDES is usually viewed as the decomposition of a single DES program into logical units that can be executed concurrently on distributed processing architectures [Fuji93]. BATTLESIM is integrated with TCHSIM (Thomas C Hartrum Simulation) which is a collection of general DES mechanisms and services and with SPECTRUM which is a parallel simulation protocol testbed based on the LP (Logical Process) model.

2.4.3.1 BATTLESIM Architecture BATTLESIM is a battlefield simulation PDES application supported by some lower-level components. Figure 2.6 [Hill94] shows the major BATTLESIM, TCHSIM, and SPECTRUM components in the context of the general DES architecture. The lowest level in the component is the AFIT version of SPECTRUM that provides the host machine process management and communication services. The SPECTRUM supports the TCHSIM as it manages the creation and communication between each LP. The processing which supports TCHSIM contains both an input/output message handler and synchronization logic for inter-LP message flow control. TCHSIM is an object-oriented collection of structures and operations that provide basic simulation services. It also is simulation kernel, providing a simulation clock, a NEQ (Next Event Queue), a top-level event dispatcher, and structural definitions for simulation events and relationship mappings [Hill94].

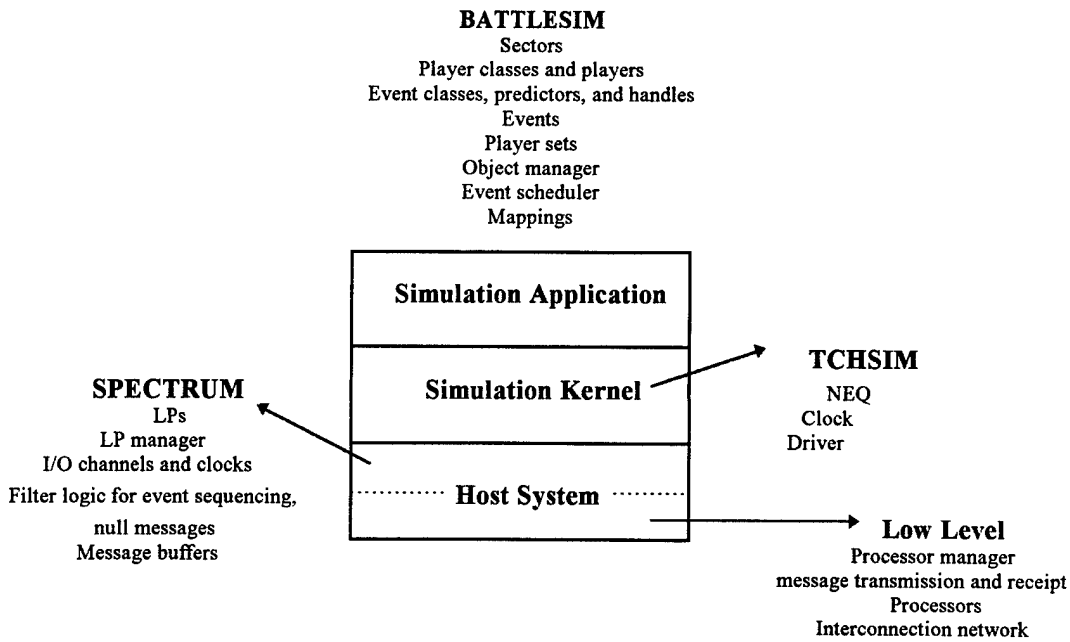

Figure 2.6 BATTLESIM, TCHSIM, and SPECTRUM

2.4.3.2 Scenario File Organization A scenario file of BATTLESIM generally consists of two parts. The first part is a Header part which declares global data for the scenario file. The second part is the Entity part that can be repeated many times. Table 2.3 shows the organization of a scenario file. In the Entity part, there are several other repeated Entity parts, as many as defined by the number of each part.

Table 2.3 BATTLESIM Input File

Name	Detailed Contents	
Header Part	Version Number	
	Terrain	File Name Minimum, Maximum Coordinates
	Sector	Number of Sectors Describing Sectors (can be repeated)
	Icon	Number of Icons Describing Icons (can be repeated)
	Object	Type, ID, Location, Velocity, Orientation, Limitations
Entity Part	Route	Number of Route Points Describing Route Points (can be repeated)
	Sensors	Number of Sensors Describing Sensors (can be repeated)
	Armaments	Number of Armaments Describing Armaments (can be repeated)
	Targets	Number of Targets Describing Targets (can be repeated)
	Defensive System	Number of Defensive Systems Describing Defensive Systems(can be repeated)

2.5 Summary

This chapter discussed the background on Distributed Interactive Simulation (DIS) and a new technology area, Advance Distributed Simulation (ADS), as well as Computer Generated Forces. It also provided a brief synopses of some SAF simulations which were recently developed and available for experimentation at AFIT. Specifically, this chapter addressed how each simulation works and what are the differences among them. It is important to first understand the architecture of each simulation and the organization of their scenario files before we can design a process for translating between these formats.

III. System Requirements and Design

3.1 Introduction

This chapter describes the requirements for the Scenario Translator and their incorporation into the system design. I specified the definition of several essential words in my thesis work area, and delineated General objectives of the system design in the requirements section. Design details follow in the design section. General requirements are introduced in the discussion of the definition phase. The overall requirements formed the basis for system design, while the detailed requirements translated into specific system capabilities.

3.2 Definitions

We use the following definitions throughout this document as shown below:

- Simulation: a method for implementing a model over time
- Scenario: a sequence of events or a possible course of action
- Scenario File: a file which stores a scenario
- Source Scenario: a scenario file which will be translated to another format
- Target Scenario: a scenario file which is produced by a translation tool
- Source Simulation: a computer simulation which operates a source scenario
- Target Simulation: a computer simulation which operates a target scenario
- Transitional Prototype (TP): an intermediate prototype to translate a source scenario to a target scenario
- Transitional Source Scenario Prototype: a data structure in computer memory which stores a source scenario
- Transitional Target Scenario Prototype: a data structure in computer memory which stores a target scenario
- Scenario File Translator (SFT): a translation program that converts a source scenario to a target scenario

3.3 Requirements

The requirements are divided into five main areas:

- Defining a Transitional Prototype,
- Loading a source scenario of a source simulation
- Mapping the source scenario to the TP,
- Remapping the TP to a transitional target scenario prototype, and
- Saving the transitional target scenario prototype as a target scenario for a target simulation.

The first requirement, defining a TP, is the most important part of my work because a good prototype can accommodate common simulations. This makes the system extensible and reusable. The second requirement, loading a source scenario of a source simulation, stems from the necessity to convert a battle scenario to one which can be run by another simulation. The third goal, mapping the source scenario to the TP, is also important since it determines how much information can be translated from the original battle scenario to another simulation. Fourth, remapping the TP to a transitional target scenario prototype is a necessary step to convert a source scenario to a target scenario. Finally, saving a transitional target scenario prototype is significant since all my effort is worthless if the target simulation cannot read the target scenario and work fine.

3.3.1 Defining a Transitional Prototype

3.3.1.1 Insufficiency of DIS PDU As mentioned previously, creating a well-defined prototype is the most important part of this work. Initially, it was desired to select a widely-used prototype. Such a selection would provide better accuracy in translating simulation

scenarios as well as save time by precluding the necessity of defining a new prototype. For these reasons, it was decided to use DIS PDUs to define the TP, since most simulations either support or will support that protocol. DIS PDUs send current events over the network, so other heterogeneous simulations can reorganize the scenario by receiving these PDUs. It also provides for the simulation to predict the direction of an entity by using the dead reckoning algorithms. Unfortunately, DIS PDUs do not provide certain critical pieces of information including the first position of an entity, amount of munitions, mission, etc. Thus, although it is possible to ascertain where an entity is going, it is not possible to determine where it came from, why it is on its current heading, or what its actions will be when it arrives at its objective. Table 3.1 provides comparison between the type of information which each simulation has and its matching data in DIS PDUs.

Table 3.1 Matching Comparison between Simulation Scenarios and DIS

Simulation Name	Number of Variables			DIS Matching Number	Percentage
	Header	Entity	Subtotal		
ModSAF	11	271	282	17	6.0
EADSIM	167	215	382	13	3.4
BATTLESIM	15	51	66	17	25.8
Total	193	537	730	47	6.4

3.3.1.2 Defining a New Prototype The previously-mentioned limitations of DIS PDUs required defining a new prototype which contains the static and scenario-wide information missing in the DIS PDUs. In order to develop a new prototype, it is first necessary to define those piece of information which form the basic scenario. First of all, the scenario entity and the mission to be performed by the entity are the basic compositions of a scenario. There are

thousands of different entity types. For example, an entity can be defined as a platoon or as ten army divisions. It could be something as small as an anti-personnel mine or as large as an airplane, tank, missile base, or aircraft carrier. An entity also has its own initial position, mission, armed capacity, orientation and velocity as well as entity ID and entity name. Items such as entity ID are especially important because they allow for unique identification. The capacity of their armament is also quite different, just as the amount and kind of munitions that can be loaded on an F-16 may vary. However, the entities themselves do not comprise the entire scenario. Only after each entity is given a mission is a scenario brought to life. Missions also depend on the types of entities. Missions for ground units are quite different than those for air units. Thus, both entity and mission must be described when defining a new prototype. Furthermore, the effect of the environment cannot be neglected in composing a scenario. Mountains, buildings and weather conditions must be considered since both sides are not fighting on a plain where everybody can see each other directly. Finally, not-readily-quantifiable items such as willingness to fight, morale, and experience can also affect a scenario's outcome and are worth consideration. A more detailed discussion of this topic appears in Section 3.3.3, Transitional File Specification.

3.3.2 Loading a Source Scenario The first step to convert one battle scenario to another is to read the source scenario. This is accomplished in most simulations via embedded procedures. Since the objective of this work was to develop a translator program, it was necessary to separate the scenario initialization code that simply reads the source scenario. This has the benefit of greatly simplifying the SFT's reader. It is reasonable that the resulting data by the loading system should be same as the one by a distributed simulation. In chapter Four, Implementation, I discuss advantages and disadvantages with using the source

simulation's reading module directly or creating a new simple reading module to load a scenario.

3.3.3 Mapping to a Transitional Prototype As described in Section 1.3, using the intermediate format is efficient because it is not dependent on the number of existing simulations. As we can see in Figure 3.1, only two routines are needed even when a new simulation is developed.

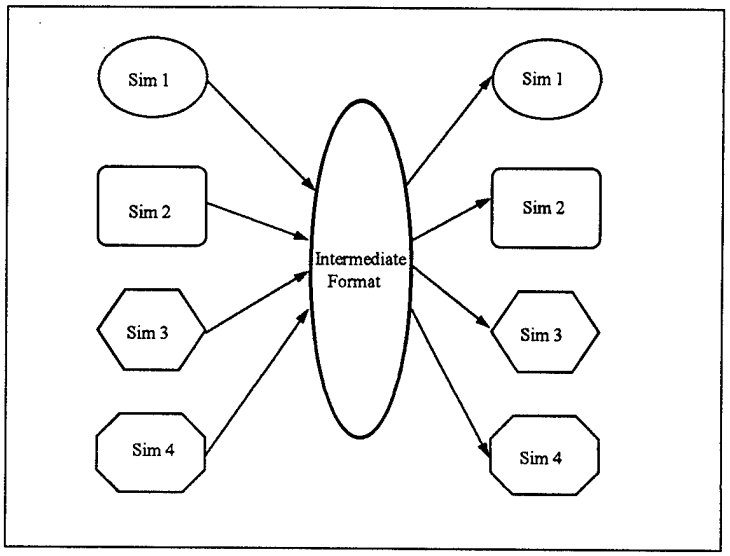

Figure 3.1 Conversion using an Intermediate Format

Matching each simulation to the TP guarantees greater extensibility than a direct matching of each simulation to one another as mentioned in Section 1.4, Approach. When converting to the TP, it is important that the mapping should be performed to the most general and acceptable data which can be remapped to any simulation. The position of an entity is a good example. The position of an entity can vary between simulations, because of terrain differences and relative coordinates used by each simulation. If the coordinates of the entity

are saved as they are when mapping to the TP, it may cause confusion in other simulations. Additionally, unless the defined TP covers all of the information contained in a source scenario, parts of the scenario may become lost.

For example, there are two simulations, each of which has a different purpose. Denoting the scenario files associated with each simulation as ***S1*** and ***S2***, respectively, we want to translate ***S1*** to ***S2***. ***S1*** is therefore a source scenario while ***S2*** is a target scenario. After mapping ***S1*** to the TP, the mapped area is shown by in Figure 3.2. This results in the area shown as becoming lost. When we define the more general intermediate prototype, the mapped area (area) becomes larger and the lost data (area) becomes smaller.

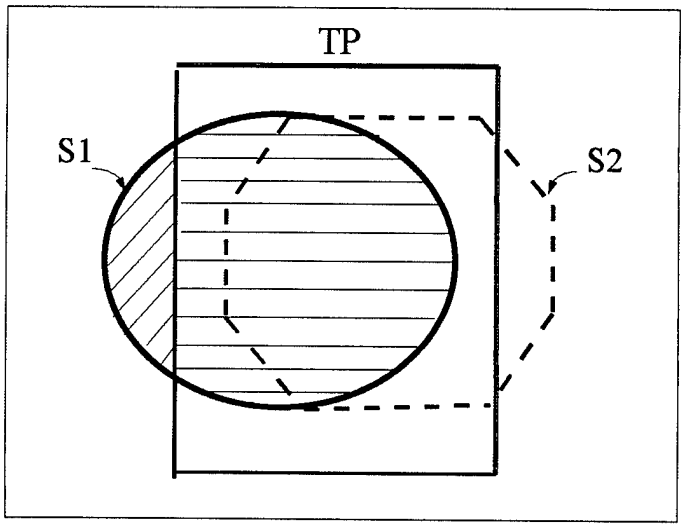

Figure 3.2 Mapped and Unmapped Area into Intermediate Format

3.3.4 Remapping to a Transitional Target Scenario Prototype After mapping a source scenario to the intermediate format, TP, it is ready to be subsequently changed to any format. When we wanted to change scenario ***S1*** to scenario ***S2*** in the previous example, it was

necessary to convert the mapped area to a writable structure to save as scenario ***S2***, in other words, to map the common area between ***S1*** and TP to a common area between TP and ***S2***. In the mapped area in the previous example, the part which is common between the intermediate format and ***S2*** is the area shown as in the Figure 3.3.

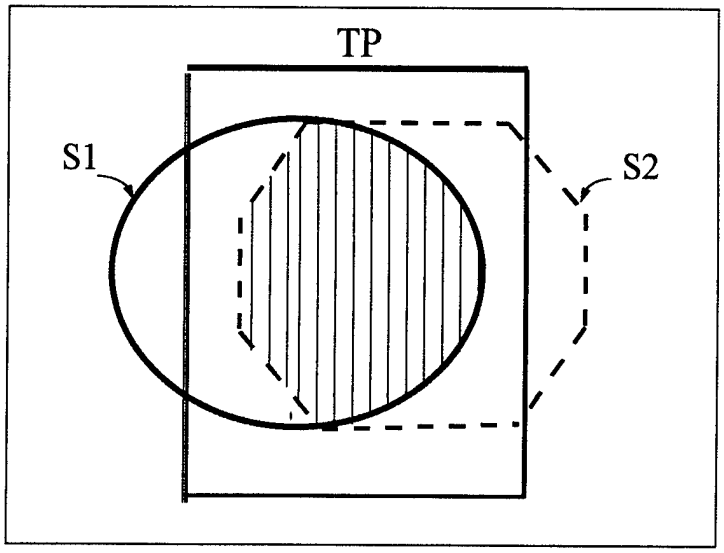

Figure 3.3 Common Area Between Intermediate Format and Target Scenario

Of prime importance in the file creation is to consider data which is defined in the TP, but cannot be re-mapped to the target scenario. In most cases, this data affects the other scenario in a limited manner since this area is information for only the target scenario such as host computer name, directory path, set-up data for background color, image type, and entity icon format. Thus, the loss of the most data in this area is acceptable. However, there still exists some critical data of a target scenario which is not represented in the source scenario. This results in lost data (area in Figure 3.4). The problem of undefined can be mitigated by mapping to commonly used default values. For example, assume a scenario needs the angle of the radar beam of an aircraft, but that information is not available. In this instance,

that information can be replaced by the common average angle of the radar beam, a value which is commonly known and can be readily approximated.

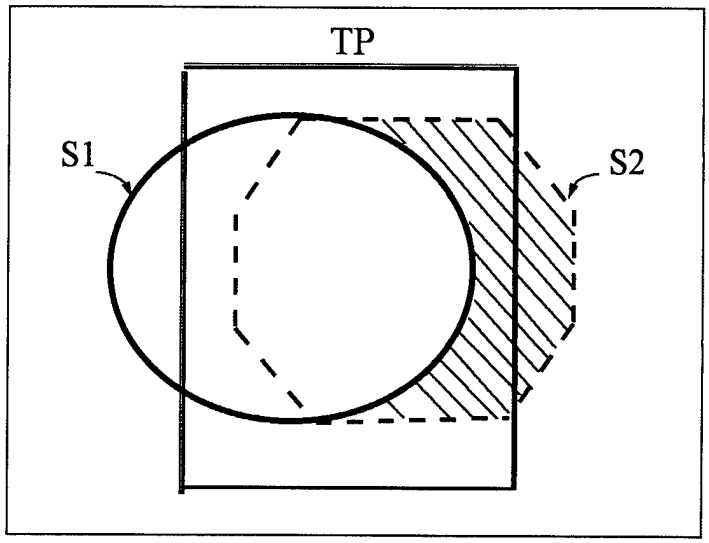

Figure 3.4 Unrepresented Area In a Target Scenario

3.3.5 Saving the Transitional Target Scenario Prototype After mapping the intermediate format, TP, to a transitional target scenario prototype, a new scenario file is created. The newly saved scenario file should be readable by the target simulation. Here we also must determine whether to use an existing output module within the target simulation's code or to develop a new module. Chapter Four discusses this issue in more detail.

3.4 Design

3.4.1 Hardware Specification I designed this program to be machine independent. Platforms used in testing were the SGI and Sun Workstations.

3.4.2 Software Specification

3.4.2.1 General Organization Figure 3.5 shows the general organization and operation process of this program. The Translator, the core of this program, loads the source scenario to be translated, maps to the TP, and then saves as a target scenario. The TP can either be stored to disk for future use or immediately converted to another target scenario.

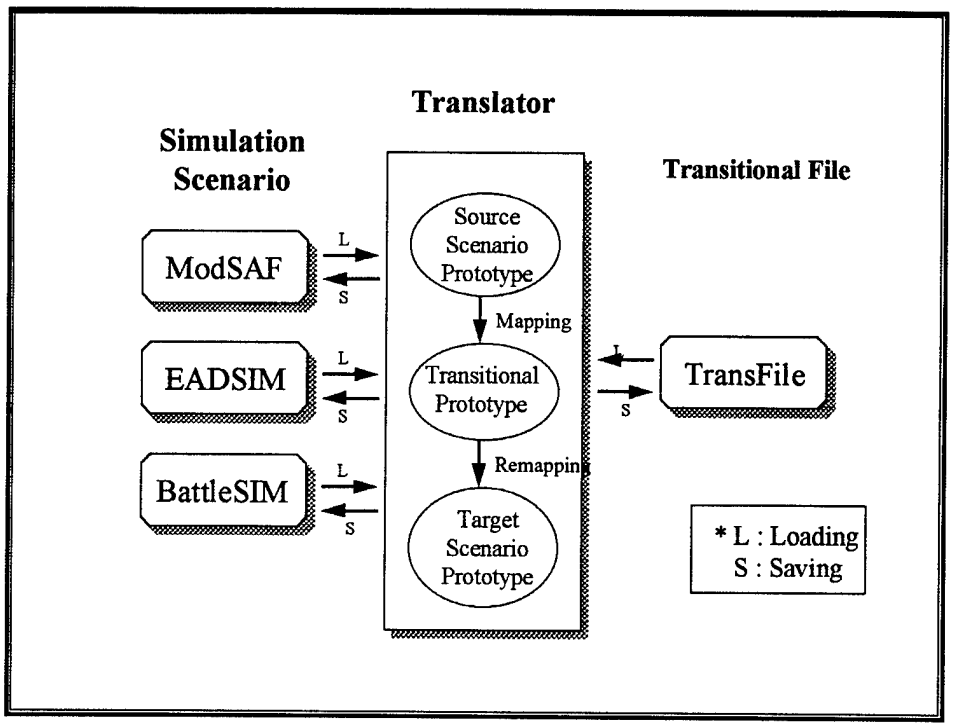

Figure 3.5 Translator General Diagram

3.4.2.2 User Interface I designed a Graphics User Interface (GUI) using the Motif library to provide a convenient interface for the user. In the instance where the Motif library is not available on the machine of implementation, I also made the program with a command line interface.

3.4.2.3 Object Diagram The Figure 3.6 shows the object diagram of the Scenario File Translator. This top-level diagram also shows the unique object classes identified for the SFT based on the decomposition of system requirements. The seven objects within the SFT are: the Translator object, the Scenario File object, the Prototype object, the Transitional File object, the Entity object, the Character object, and the Mission object. The Simulation object is composed of the Scenario File object, the Terrain object, and the Initial Setting object. Since the SFT only interfaces with the Scenario File object, only this object is important to my design. The Scenario File object consists of the Entity objects, which have their own characters and missions as mentioned in Section 3.2.1.2, Defining a New Transitional Prototype.

The Translator object contains the Prototype and Transitional File objects and uses the Scenario File object to load and save the TP. To map from the source scenario to a target scenario, the Translator object loads into memory each simulation transitional format. The reason why the transitional scenario prototype is needed is discussed in Section 3.4.4, Transitional Target Scenario Prototype. After the Translator loads the source scenario, it maps the scenario to the Prototype object. The Translator can save the scenario as either the target scenario or the Transitional File format.

To save to the target scenario, two steps are needed. The first step is to map the TP to the transitional target scenario prototype, then the Translator program saves the TP as a target scenario, which the target simulation can then use.

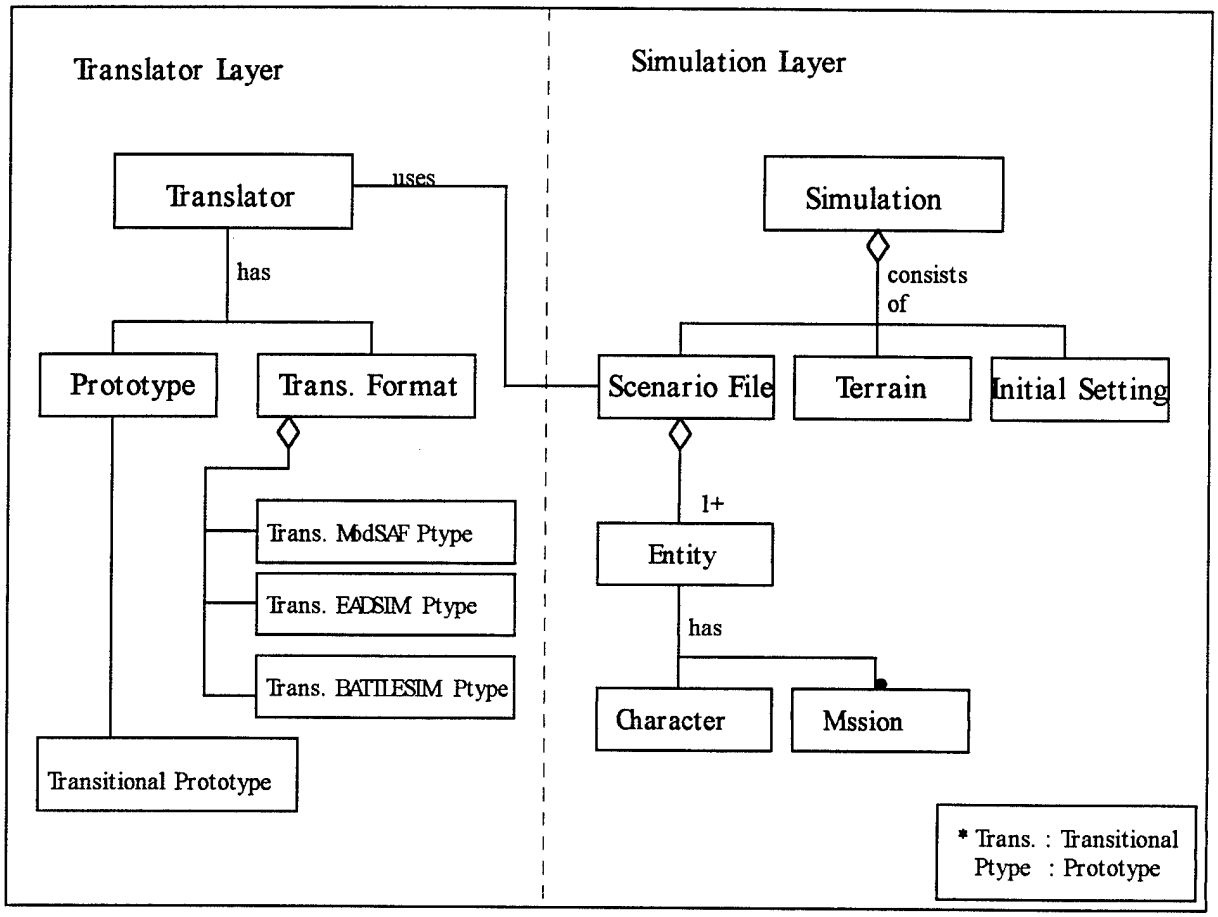

Figure 3.6 Scenario Translator Object Diagram

3.4.3 Transitional Prototype Requirements

3.4.3.1 Ideal Prototype The ideal format for the TP is a set of information which covers all data for all simulations including those which are not currently supported by the translator. Clearly, this is not practical; therefore, including parts generally common to all simulation scenarios is advisable.

3.4.3.2 Proposed Prototype The basic essential composition of a scenario, the characteristics and missions of the entities, have to be describable in any format I propose. Extensibility of the translator is promoted when the way to address the entity and mission is

accomplished through general methods. Figure 3.7 shows the relationship among ModSAF, EADSIM, BATTLESIM, and the TP.

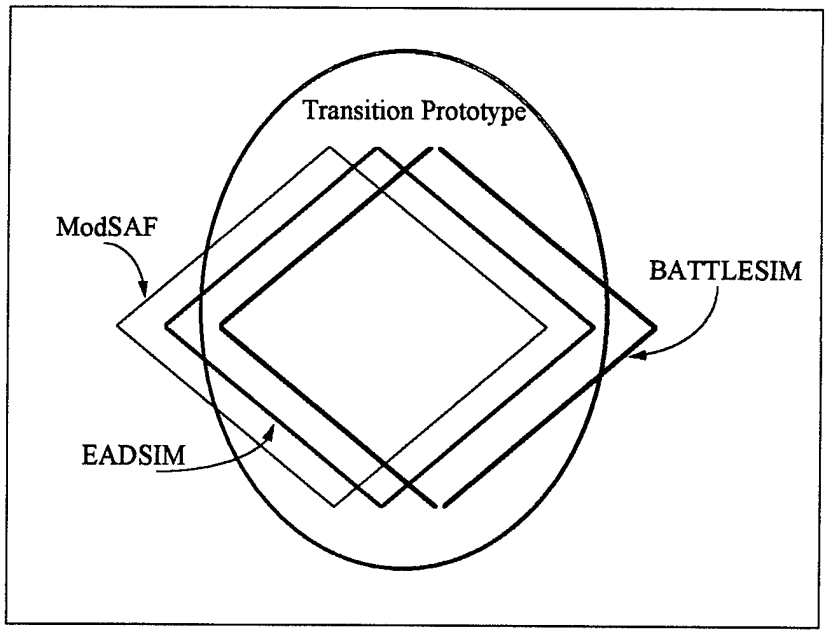

Figure 3.7 Relationship between Simulations and Suggested Prototype

The overlapping parts in the Figure 3.7 indicate the common parts between any two simulations. The size of each rhombus and a circle in the figure does not present the actual file size of each scenario and the intermediate format, respectively. The center part where the three rhombuses overlap is the most common information which is present in all three simulations. As is evident in the figure, some data can be lost in mapping from a scenario into the TP. In this case the default values should be matched to the undefined part in the TP during mapping from the TP to a target scenario. How I accomplished this is discussed in the following section.

3.4.4 Transitional Scenario Prototype Specification

When loading a source scenario, it is desirable to read all information contained in the using source simulation's original data structures. Thus the transitional source scenario prototype are designed to read all information whether needed by another simulation or not to minimize data loss. The rest of this section describes the data structure of the source scenarios and tells how these structures influenced the design of the transitional data structure. The reader may refer to Appendix A for more details about each transitional source scenario prototype.

3.4.4.1 ModSAF As mentioned in section 2.4.1.2, a ModSAF scenario file is divided into two sections: a Header part and a Class part. Figure 3.8 illustrates the top-level data structure of a ModSAF scenario file.

Transitional ModSAF Scenario Prototype

Pointer to Header Part
Array Pointer to File Entry
Array Pointer to Class part

Figure 3.8 Transitional ModSAF Scenario Prototype

3.4.4.2 EADSIM It is not possible to create a scenario by reading only one file in EADSIM because a complete scenario is contained in several different files. The important files among them are the laydown file which stores the platforms describing each entity, and the scenario file which describes the initial conditions and the name of laydowns. Figure 3.9, Figure 3.10 and Figure 3.11 shows the Prototypes for a 'scenario file', a 'laydown file', and a platform.

Transitional EADSIM 'Scenario File' Prototype

Structure of Header
Pointer to Laydown File Name
Structure of Element Map
Pointer to Network File
Structure of Zero thru Max Host
Structure of Unused String
Structure of Debug
Structure of Host
Structure of Max File thru C3I Output
Structure of Truth
Structure of SPDS
Structure of Log
Structure of Statistics
Structure of Probability Statics
Structure of File
Structure of Earth Rad
Structure of Hzulu
Structure of Display thru Route
Structure of Format thru Starting
Structure of Processing
Structure of Write
Structure of C3I Log
Structure of Image
Structure of Trans Rad File Path
Structure of Time
Structure of External
Structure of TBA

Figure 3.9 Transitional EADSIM 'Scenario File' Prototype

Transitional EADSIM 'Laydown File' Prototype

Version Number
Number of Platforms
Pointer to Platform

Figure 3.10 Transitional EADSIM 'Laydown File' Prototype

Transitional EADSIM Platform Prototype

Military ID
System ID
Commander ID
Color Index
Pointer to Sensor
Number of Communicaitons Devices
Pointer to Com Dev
Pointer to Jammer
Number of Assests
Pointer to Asset
Number of Targets
Pointer to Target
Number of IFTU Platforms
Pointer to IFTU Platform
Boolean of Sim, Hhour, Local, Zulu, HM No Delim, HMS, As Entered
String of Time Suffix
Boolean of Log Track Event, Laser Absolute, Laser Rest Az, laser Rest El
Platform Type
Structure of Satellite
Boolean of Use Route Data
Pointer to Use Route Data
Pointer to Not Use Route Data
Number of PTLs
Pointer to PTL
Number of LCS
Pointer to LCS
Number of Decoys
Pointer to Decoys
Number of Surv Platform
Pointer to Surv Platform
Boolean of CMData, EM Com Default, EM Con Authority
Pointer to Aircraft
Pointer to Airbase

Figure 3.11 Transitional EADSIM Platform Prototype

3.4.4.3 BATTLESIM It is sufficient to read one file for loading a BATTLESIM scenario. Figure 3.12 depicts the Prototype for a BATTLESIM scenario.

Transitional BATTLESIM Prototype

Object Type
Object ID
Current Time
Update Origin
Number of Events
Structure of Location
Structure of Velocity
Structure of Orientation
Structure of Rotation
Player Size
Mass
Pointer to Polygon List
Pointer to Subclass

Figure 3.12 Transitional BATTLESIM Prototype

3.4.4.4 Transitional Prototype Table 3.2 shows a data structure of the TP. Each structure consists of variables that have same character. The column 'Type' describes the variable format of the structure, and the enumeration type is described in Appendix B. The variable name 'Number of Entities' indicates how many entities are contained in the transitional file. The remainder of the file contains specific information about each entity. The entity field structure consists of nine subfields: Entity Header, Entity Type, Entity Location, Entity Velocity, Entity Orientation, Munitions, Sensor, Jammer, and Task field. Each entity is uniquely identified by the number contained in the Entity Header field. The Entity ID should be unique in a scenario. The Force ID in the Entity Header structure indicates whether the entity is a friendly or enemy force.

If the entity has a name, the Entity Name field in the Entity Header stores it. The Entity Type Structure shows the kind of the entity. It follows the DIS standard as specified in the document, *Enumeration and Bit-encoded Values for use with IEEE 1278.1-1994, Distributed Interactive Simulation - Application Protocols* [IST94-1]. I designed for

Table 3.2 Data Structure of Transitional File

Name		Type
Structure Name	Variable Name	
Number of Entities		Integer
for each entity		
Entity Header	Entity ID	Integer
	Force ID	Integer - enumeration
	Entity Name	Character String
Entity Type	Domain	Integer - enumeration
	Country	Integer - enumeration
	Category	Integer - enumeration
	SubCategory	Integer - enumeration
	Specific	Integer - enumeration
Entity Location	Location X	Float
	Location Y	Float
	Location Z	Float
Entity Velocity	Velocity X	Float (meter/sec)
	Velocity Y	Float (meter/sec)
	Velocity Z	Float (meter/sec)
Entity Orientation	Psi	Float
	Theta	Float
	Phi	Float
Number of Kind of Munitions		Integer
for each munition		
Muntion Type	Domain	Integer - enumeration
	Country	Integer - enumeration
	Category	Integer - enumeration
	SubCategory	Integer - enumeration
	Specific	Integer - enumeration
	Amount	Integer
Number of Sensors		Integer
for each sensor		
Sensor ID		Character String
Number of Jammers		Integer
for each jammer		
Jammer ID		Character String
Number of Tasks		Integer
for each task		
Task Type		Integer - enumeration
Repeated as declared in Number of Entities		

program extensibility by saving in DIS format, and a new entity type can be added by referencing the DIS Enumeration Document. The Entity Location Structure shows the initial

position of the entity. For global use of the TP, this position must be converted to a global position. It is also designed for program extensibility using the coordinates system which DIS uses. The Entity Velocity Structure indicates the initial speed of the entity in meter/sec. The Entity Orientation contains the entity's direction vector in 3D space. Entity Velocity and Entity Orientation also use the DIS format. The Number of Kind of Munitions field shows the munitions load of the entity. For example, an F-16 can carry six US Mavericks, two US Sidewinders, 15 US Mk82s, and 2,000 US M50s. In this case the Number of Kind of Munitions field is four. The Munitions Type also follows IST94-1. Some entities have sensors and/or jammers for radio communication. That information is contained in the Sensor, and Jammer fields, respectively. The Task field indicates how many tasks the entity will perform. There can exist numerous missions depending on the specific characteristics of each simulation. I adopted the task types which ModSAF simulation currently uses for an initial implementation. Interested readers can refer to Appendix C for more detail on each field. The Number of Entities in the transitional file is given by the 'Number of Entities' entry declared on the first line. In the case where the number of items is initially unknown, memory requirements are minimized through dynamic memory allocation.

3.4.5 Translation Tables

The next several figures and table represent the scenario file data from each scenario source and how this data is mapped into the TP. The left–hand side of the Table 3.3 shows the number of variables that are in common area between two scenario in each simulation; the right–hand side of the table shows how many variables are defined in the TP. Figure 3.13, 3.14, and 3.15 depict the mapping of the scenario data from ModSAF, EADSIM, or BATTLESIM, respectively, into the TP. These figures use the same notation as in [Gard93].

Table 3.3 Percentages for Common Variables between Two Scenario

Simulation to Simulation	Number of Common Variables	Number of Common Variables defined in TP	Percentage(%)
ModSAF to EADSIM	18	18	100
ModSAF to BATTLESIM	21	21	100
EADSIM to BATTLESIM	24	24	100

3.5 Summary

This chapter defines the requirements for converting one scenario file to another. These requirements can be grouped into five main areas: first, defining a TP; second, loading a source scenario of a source simulation; third, mapping the source scenario to the TP; fourth, remapping the TP to a transitional target scenario prototype, and finally, saving it as a target scenario for a target simulation. Specific implementation details of the goals are discussed in the next chapter.

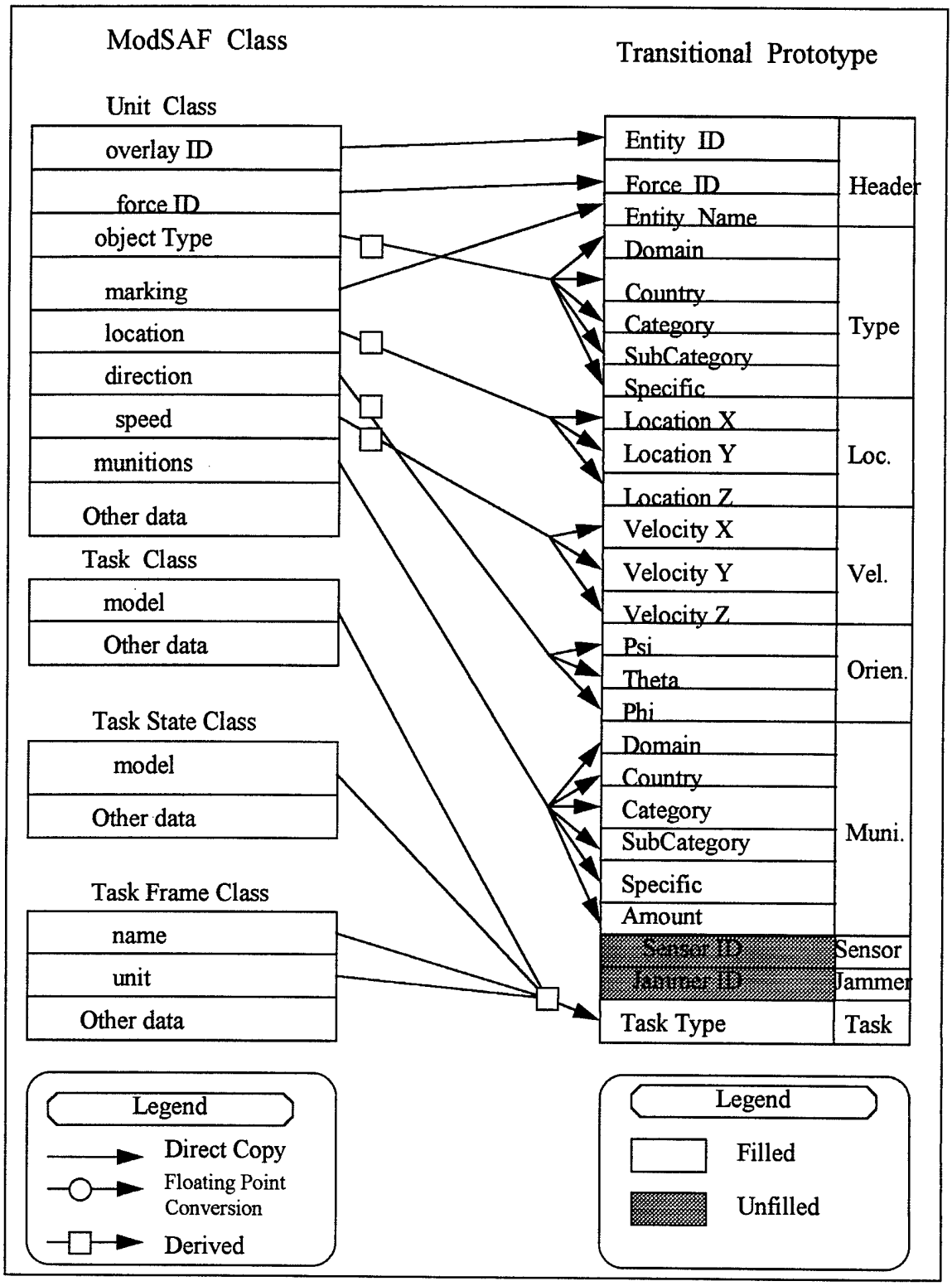

Figure 3.13 Transitional Prototype mapping for ModSAF scenario

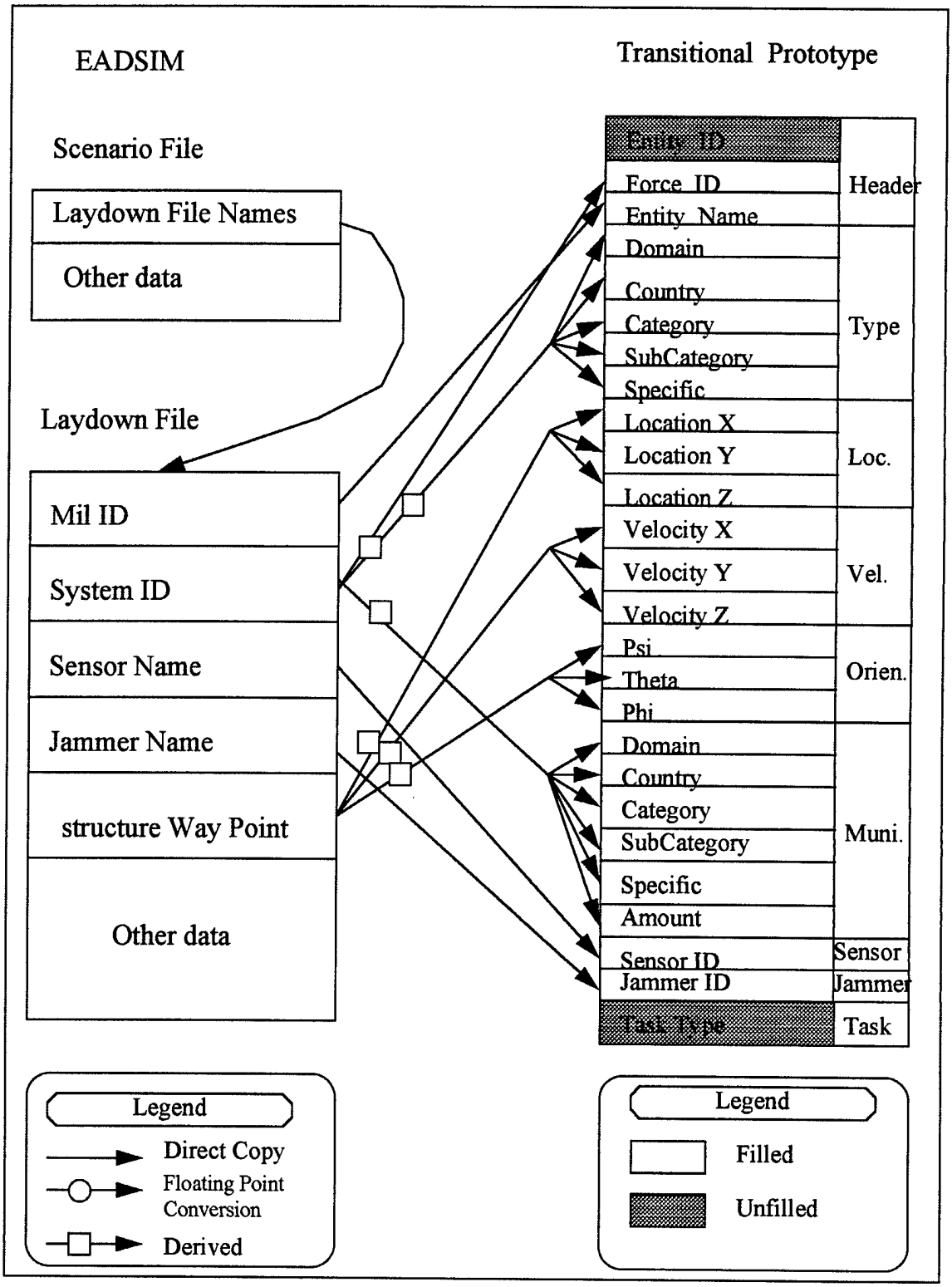

Figure 3.14 Transitional Prototype mapping for EADSIM scenario

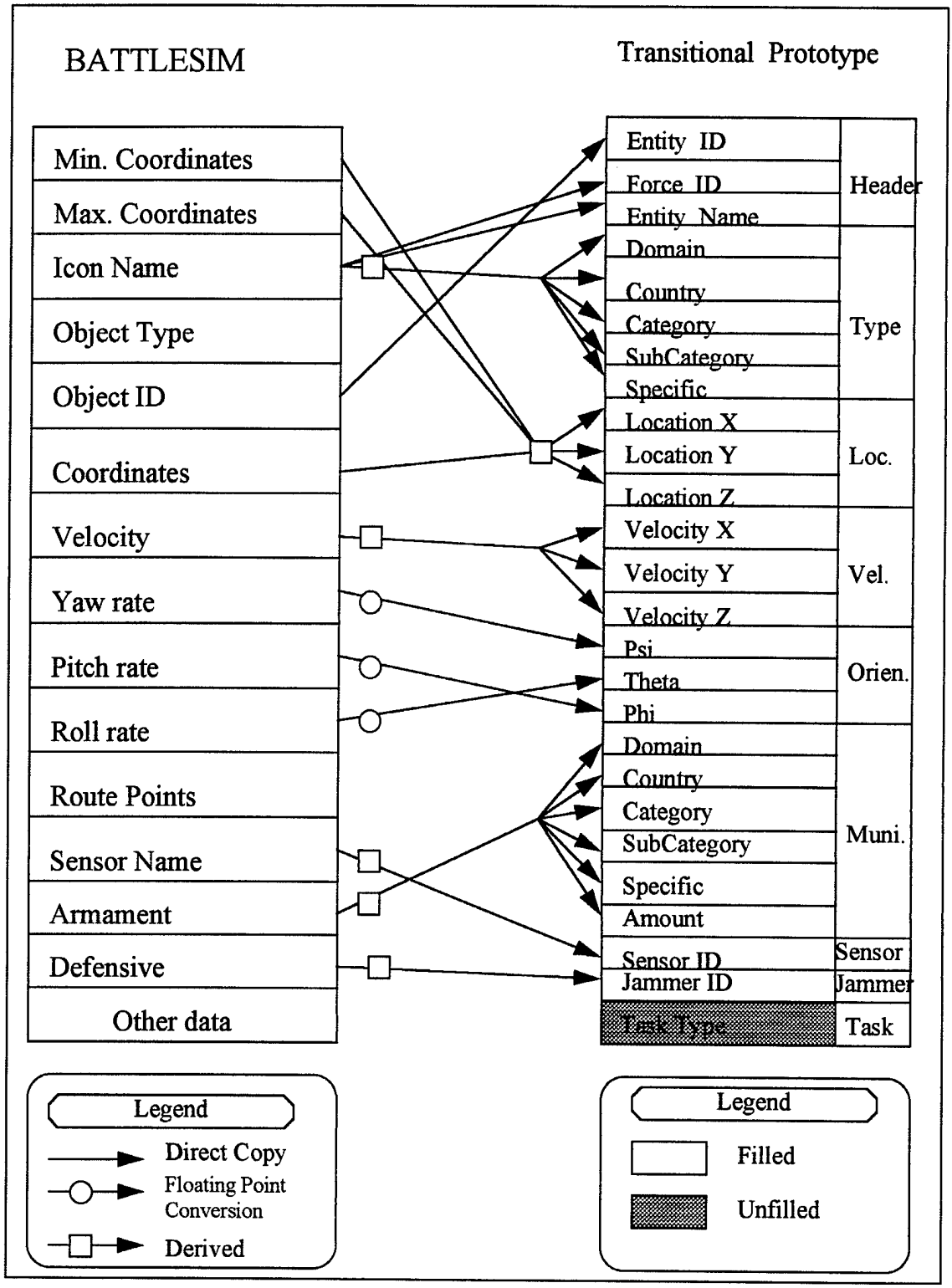

Figure 3.15 Transitional Prototype mapping for BATTLESIM scenario

IV. System Implementation

4.1 Introduction

This chapter discusses the implementation of the Scenario File Translator (SFT)'s design defined in Chapter 3. This implementation utilizes five subsystems: a loading system, a mapping system, a remapping system, a saving system, and a user interface system. The loading system reads a source scenario and stores it to a source scenario prototype. The mapping system transforms the transitional source scenario prototype to a Transitional Prototype (TP). The remapping system transforms the TP once more into a transitional target scenario prototype. The saving system writes the transitional target scenario prototype as a scenario file which can then be run by the target simulation. Finally, the user interface system provides a Graphics User Interface (GUI) to this program easier to run. All programs are written in the C programming language and were tested on SGI and Sun Workstations.

4.2 Loading a Scenario File

Reading a scenario file stores the scenario in computer memory. For this to work, a data structure which defines the organization of the scenario is needed. This structure is called the transitional source scenario prototype. Since the SFT operates three simulations' scenario types, this section discusses the loading system for each of the three scenarios.

4.2.1 Loading a ModSAF Scenario File There are two ways to load a ModSAF scenario file. The first way is to use built-in libraries in the ModSAF program which do more than simply read the file and the other are to design and develop a new function to only read the scenario file. Each method has its own advantages and disadvantages.

First, using the ModSAF built-in libraries, we save time since we do not need to design and implement new functions; however, it is necessary to include many unnecessary files even

though a large number of unused functions may be contained in those files. This results in excessive compilation requirements. This method is inefficient, since the loading system of the SFT needs just one routine to read a scenario file. Conversely, designing and implementing a new function is attractive, because it simplifies the general organization of the program. On the other hand, this method requires additional work for designing a scenario data structure to implement the loading system. Since program readability and understandability are more important, I chose to design and implement a new library.

As mentioned in Section 2.4.1.2, the structure of a ModSAF scenario file consists of two parts: the Header part and the Entity part. The Header part describes the scenario organization, and the Entity part describes the entities in the file. The Entity part is also divided into the File Entry part and the Class part. The File Entry part has basic values for scenario class, and the Class part describes all about the class. The most important information in the Header part is the number of entities in the scenario file. The ModSAF loading system reads classes individually. The number of classes read is given by the number of entities. In the File Entry part, the object type and the variant size of the class is also important. Moreover, it is necessary to combine the classes with their related entities. Once the scenario file has been read, it is stored in the transitional prototype data structure for ModSAF scenarios. The minimum memory requirement to read a ModSAF scenario file with varying number of entities is described in Table 4.1. Examples shown are for cases where each entity has just one task.

4.2.2 Loading an EADSIM Scenario File Because EADSIM scenarios are saved in a text format, it is relatively easy to understand their organization and to develop an internal data structure for EADSIM scenario file information.

Two primary files are needed to build an EADSIM scenario: the scenario file and the laydown file. The laydown file stores all the information about the entities which compose a scenario, and the scenario file describes initial setup conditions and references the related laydown

Table 4.1 Required Memory Size for Loading a ModSAF Scenario File

(* B : Bytes)

Entity Number	Header Part	Number of Class	Entity Part		Total (KBytes)
			File Entry Part	Class Part	
1	72 B	30	360 B	3,764 B	4.1
100	72 B	3,000	36,000 B	376,400 B	405.7
1,000	72 B	30,000	360,000 B	3,764,000 B	4056.7
10,000	72 B	300,000	3,600,000 B	37,640,000 B	40,556.4

file names. Additional files are required to organize a scenario; however, these two former files are more important than the others.

To load an EADSIM scenario, you must know which laydown files are connected to a scenario. The laydown files are read in order to get the entity information composing the scenario. Once this data is stored in the transitional EADSIM source prototype as defined in Chapter 3, then it is possible to progress to the next phase, the mapping phase. This stage is discussed in the Section 4.3.

Loading an EADSIM scenario is straightforward; just read the text files into the transitional EADSIM source prototype data structure. In the case of a scenario file, only the laydown file names in this file are important since the remainder of its data is only used for building the EADSIM scenario itself. However, the loading system reads all the data in an EADSIM scenario file for further use since currently unused data may be useful later.

In the case of EADSIM, the required memory size of a typical scenario file including laydown file names, with one entity is about 8.3 Kbytes. As with ModSAF, EADSIM scenario file size is dependent on the number of entities. Table 4.2 shows typical cases for scenarios with varying number of entities.

Table 4.2 Required Memory Size for Loading an EADSIM Scenario File

(* B : Bytes)

Entity Number	Scenario File	Laydown File	Total (KBytes)
1	7,509 B	988 B	8.3
100	7,509 B	98,000 B	103.8
1,000	7,509 B	988,000 B	972.2
10,000	7,509 B	9,880,000 B	9,655.7

4.2.3 Loading a BATTLESIM Scenario File It is also fairly straightforward to load a BATTLESIM scenario file, since it is also saved as a text file like an EADSIM scenario file. To load a BATTLESIM scenario file, the data is extracted from each line in the source file into the transitional BATTLESIM source prototype. When reading a data line from the source scenario file, if the first character of the line is an asterisk defined by the programmer, it skips the line and reads the next line automatically so that some comments can be written in the scenario file for providing better understanding. Figure 4.1 shows this algorithm.

```
char        Data_Line[MAX_LINE_LENGTH];
while (!feof(File_Ptr))             /* read until end of file */
{
  fgets (Data_Line, MAX_LINE_LENGTH, File_Ptr);
  switch (Data_Line[0])
  {
    case '*':               /* ignore all lines that starts with characters */
    case '\n':              /* '*', '\n', '\0', or space                    */
    case '\0':
    case ' ':
      break;
    default:
      return Good_Line;     /* Boolean value */
    }                       /* end of switch */
}                           /* end of while */
```

Figure 4.1 Algorithm for Reading a BATTLESIM Scenario file

Assuming an entity has only one route point, sensor, armament, target, and defensive system in a BATTLESIM scenario, then 276 Bytes are needed for a scenario with one entity. Table 4.3 describes these memory requirements for similar scenarios with different numbers of entities..

Table 4.3 Required Memory Size for Loading a BATTLESIM Scenario File

Entity Number	Header Part	Entity Part	Total (KBytes)
1	144 B	132 B	0.27
100	144 B	13,200 B	13.0
1,000	144 B	132,000 B	129.1
10,000	144 B	1,320,000 B	1,289.2

4.2.4 Loading a Transitional File As will be seen in Section 4.5.4, a Transitional File is also saved in text format so it can be easily verified when viewing the components in it. A Transitional File is read and loaded into the Transitional Prototype data structure using the same algorithm for loading a BATTLESIM scenario file.

The required memory size for each entity in the TP depends on the entity's number of tasks. If an entity has an 'Assault' task (the largest task with respect to memory requirements), the entity needs 271 Bytes. This is shown in Table 4.4 with other examples.

Table 4.4 Required Memory Size for Loading Transitional File

(* B : Bytes)

Entity Number	Without Task	Task	Total (KBytes)
1	218 B	53 B	0.26
100	21,800 B	5,300 B	26.4
1,000	218,000 B	53,000 B	264.6
10,000	2,180,000 B	530,000 B	2,646.4

4.3 Mapping a Scenario Prototype to a Transitional Prototype

Mapping to a Transitional Prototype (TP) is necessary to convert a source scenario to a target scenario. The more source types we map to the TP, the more data we remap to target scenario prototype and the more we increase the conversion percentage. This results in a more well-translated target scenario. Because the SFT uses scenarios from three simulations, six mapping routines are needed. This is because each simulation needs two mapping routines: one to the TP from the transitional source scenario prototype, and another from the transitional target scenario prototype to the TP. This section discusses mapping a transitional source scenario prototype to the TP. Section 4.4 discusses remapping the TP to a transitional target scenario prototype.

4.3.1 Mapping a ModSAF Prototype to a Transitional Prototype A ModSAF scenario file is stored internally as a transitional ModSAF prototype as defined in Section 3. Mapping from this structure to the TP is done by traversing the stored structure. Ten of the 21 ModSAF classes are essential for mapping to the TP. They are the Overlay Class, the Point Class, the Line Class, the Sector Class, the Unit Class, the Task Class, the Task State Class, the Task Frame Class, the Task Authorization Class, and the Minefield Class. Among them, four classes are more useful for mapping to the TP (see Figure 3.13). To begin, it is necessary to determine how many entities there are in a scenario file. The number of the entities is same as the Unit Class number in the ModSAF scenario file. If the number of the entities in a scenario file is 50, then the Unit Class number is also 50. Thus, if Unit Class number is determined while traversing the transitional ModSAF prototype, the number of the entities in the TP is found easily. Since the Unit Class has direct information of Entity ID, Force ID, and Entity Name in the TP, every time a Unit Class is detected they are mapped to the TP using these items. The TP Entity Type is stored in the DIS format which has Domain, Country, Category, SubCategory, and Specific values. The variable 'objectType' in a Unit Class is related to the DIS format by a ModSAF function which converts this variable to the DIS format.

Since ModSAF entity location uses the topocentric coordinate system in which the origin of the coordinates is given point, we cannot directly use these coordinates within the TP. It is necessary to match this location to the DIS geocentric coordinates. The origin of the geocentric coordinate system in DIS is the centroid of the earth as shown at Figure 4.2 [IST94]. ModSAF also provides a function to support this conversion.

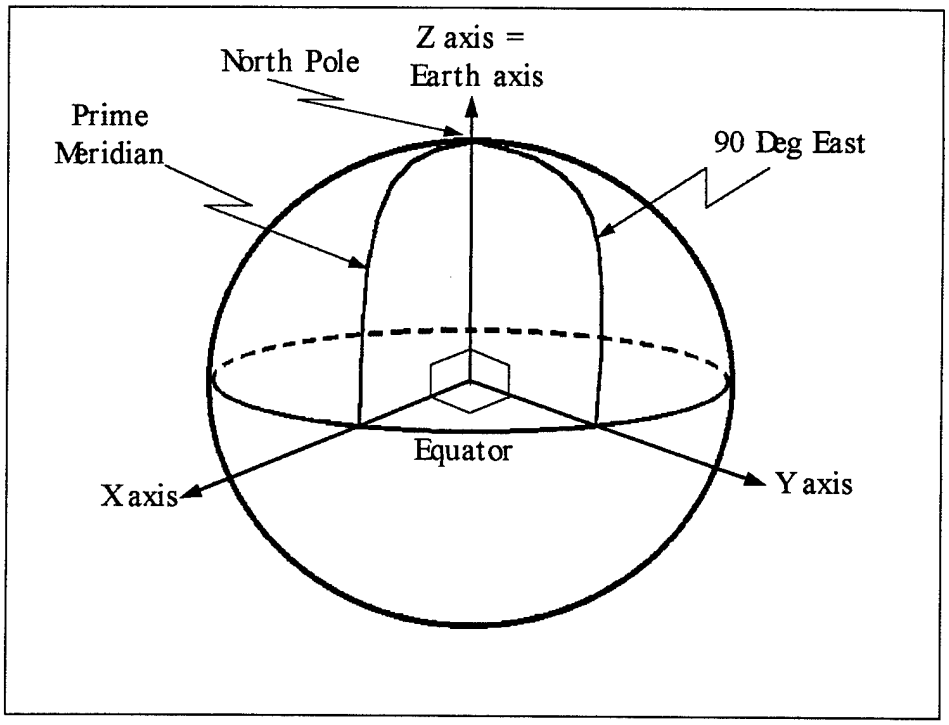

Figure 4.2 Geocentric Cartesian Coordinates in DIS

Since ModSAF must map Entity Velocity and Entity Orientation to DIS formats when it sends DIS PDUs, we can also use functions to build the TP. The kind of munitions in an entity is known from a 'munition' variable in a Unit Class of ModSAF scenario file. Another ModSAF function converts this object type to a DIS format is also used here. The amount of munition that the entity has is also stored in the 'munition' variable in a Unit Class. The Task Type in the TP can be determined after integrating several classes in the ModSAF scenario file. Some classes

which help to determine the TP Task Type are the Task Class, the Task State Class, the Task Frame Class, and the Task Authorization Class. ModSAF 2.1 defines 33 tasks, and the SFT accommodates all of them to provide a complete task mapping. Appendix B shows the task types defined within the TP.

4.3.2 Mapping an EADSIM Prototype to a Transitional Prototype Of the two data structures in the EADSIM prototype, i.e., one for the scenario file and the other for a laydown file, only the data structure corresponding to the laydown file can be directly mapped into the TP since the data in the scenario file is EADSIM specific except for the laydown file names. Specifically, five data items from the laydown data structure cover almost every part of the TP. The Entity ID and the Entity Task in the TP Entity Header are not known directly from the laydown file, so the most widely used default values are put into them. The Entity ID is produced sequentially whenever an Entity is detected, and when the entity is an aircraft the Entity Task is either 'Assault Ground Target' or 'CAS Mission' since it is required to attack all enemy forces on its path (See Figure 3.14).

We determine the Force ID in the Entity Header part, whether it is friendly or an enemy force, after checking the System ID in an EADSIM scenario file. The TP Entity Name is known when referring to the EADSIM Military ID. The TP Entity Type is saved as an enumeration type in DIS format using the EADSIM System ID. Although, there is no item in the EADSIM prototype which indicates an entity's initial location, the starting location can be determined as the first point in the path the entity passes through. This point must also be changed to a DIS geocentric coordinate system value.

We derive the TP Entity Orientation by computing the difference between the first and the second entity positions in its path. This difference is converted to the DIS format, and stored to the Entity Orientation in the TP. An entity's munition load is determined by referencing another

file in EADSIM using the System ID of the entity. This data must be converted into the DIS format. The Sensor Name and the Jammer Name are determined from the platform of the entity directly.

4.3.3 BATTLESIM Prototype to Transitional Prototype The BATTLESIM prototype consists of the Header part and the Entity part as described at Section 2.4.3.2. The items in the Header part to match to the TP include the icon definition records as well as the minimum and maximum coordinates of a terrain database. Minimum/maximum coordinates are needed when converting the entity position to geocentric coordinate system in DIS, and Icon Definition Records are for the type of the entity. The datum that is not mapped directly to the TP from a BATTLESIM prototype is the Task Type. We use the most widely accepted default values here. In case of an aircraft, we set the Task Type to either 'Assault Ground Target' or 'CAS Mission.' The task 'Assault' or 'Move Task' is used for a ground entity.

We derive the TP Entity ID from the BATTLESIM object ID. Icon Name determines the Force ID, Entity Name, and Entity Type of the TP. Entity with US icon name is recognized as friendly; Soviet icon name characterizes entity as an enemy. Otherwise, the entity is regarded as 'others.' The TP Entity Type is also determined by the icon name and translated into the DIS format. Table 4.5 gives some examples for mapping the BATTLESIM icon name to a DIS Entity name.

Table 4.5 Icon Name in BATTLESIM vs. Entity Name in DIS

Icon Name in BATTLESIM	Entity Name in DIS
f18	FA18
mig	USSR_Mig21
missile	US_CG41
tank	US_M1A1
truck	US_M35A2

The TP Entity Location is calculated using the coordinates of the entity and the minimum/maximum coordinates of the terrain file in the BATTLESIM scenario file. The TP Entity Velocity and the TP Entity Orientation are described in the BATTLESIM scenario file; however they must be converted into the DIS format. The munition type of an Entity is related to the BATTLESIM armament type; however, it cannot be used directly in the TP, but is converted to DIS format. The entity's Sensor ID and the Jammer ID are related to the Sensor Name and Defensive system in BATTLESIM.

4.4 Remapping a Transitional Prototype to a Transitional Target Scenario Prototype

In the process of converting a TP to a target scenario prototype, some items which are not represented by the target scenario are lost. In addition, it is also often necessary to generate unspecified parameters as default values in a target scenario prototype.

4.4.1 Transitional Prototype to ModSAF Scenario Prototype To make a file which ModSAF can read after remapping from a TP, it could be accomplished by a function fitting the data structure of ModSAF scenario file. I think the best approach to create the ModSAF target file is to use a routine which ModSAF provides. It is necessary to map into a Persistent Object (PO) database for saving as a scenario of ModSAF. Since much information about Semi-Autonomous Forces (SAF) behavior cannot be shared via the DIS protocol, the PO protocol was created to provide a more flexible and scaleable interface between the components of the SAF system. The PO protocol enables the sharing of behavioral states, command and control information, and system administration. ModSAF software modules have unlimited access to all the information being broadcast via this protocol, provided they are running on the same PO 'database ID.' The database ID is an identifier in the PO PDU Header that identifies with which repository of persistent objects or 'database' the application interacts.

ModSAF program provides a function 'po_create_object()' which creates an entity in the PO database. We use the function to build of each ModSAF class. Ten of 21 (see section 4.3.1) ModSAF classes are involved when remapping to a ModSAF scenario file. Among these, the Unit Class has the most important role. This class stores the general data of an entity. The matching items from the TP are 'overlayID,' 'forceID,' 'objectType,' 'marking,' 'location,' 'direction,' 'speed,' and 'munition' of the Unit Class (see Figure 4.3). The ModSAF overlayID is mapped from the TP Entity ID, and the ModSAF objectType is converted from the data that is saved in DIS format to an integer type variable through a ModSAF utility function.

The marking in the Unit Class is copied from the TP entity name. Getting the ModSAF location value requires a routine to convert the DIS format data in the TP into ModSAF terrain database. Direction, speed, and munition in the Unit Class are passed in the same way as ModSAF location data. Translating the TP task into each ModSAF Task class is required to map related data using the TP Task Type module. After accomplishing this, we build the PO_database.

4.4.2 Transitional Prototype to EADSIM Scenario Prototype In section 2.4.2.2, we mentioned that EADSIM uses two files to build a scenario. Here we describe the process of mapping from each TP entity to the EADSIM scenario prototype. The laydown file name is contained in the scenario data structure. This file contains the platform data of each EADSIM entity.

The EADSIM program allows several laydown files in a single scenario; however, the SFT uses only one laydown file to save all entities in the TP. Since this file has all the information of a scenario, we only need to open this one file when we want to see what entities are in the scenario. The EADSIM Military ID of an entity is same as the TP Entity Name. The EADSIM System ID can be obtained using the TP Entity Type. The Sensor Name and the Jammer Name in the EADSIM prototype are also derived from the TP. The TP did not provide EADSIM Way Point directly; however, we can derive the data from the TP task type. If an entity

has a task, a route which the entity should follow is stored in the TP. The Way Point can be built using these route data.

4.4.3 Transitional Prototype to BATTLESIM Scenario Prototype A BATTLESIM scenario prototype contains a Header Part and an Entity Part. The data needed to map to the Header Part is the Icon Definition part. For determining these icon definition records, it is necessary to describe the Header Part after extracting the kinds of entities which are stored in Entity Part. The BATTLESIM entity object type determines the icon number which matches with the TP Entity Type. The TP Entity ID determines the BATTLESIM object ID. The BATTLESIM coordinates are found using both the TP location and minimum/maximum coordinates as described in the Header Part.

The BATTLESIM Velocity and Orientation are derived from the TP velocity and the orientation. The sensor name and the defensive system are recognized by the Sensor ID and the Jammer ID, respectively. Finally, the armament of an entity can be derived from the TP's munition type.

The next three figures represent the transitional prototype how its data is mapped into each target scenario prototype. The size of each box is not intended to represent the actual file size.

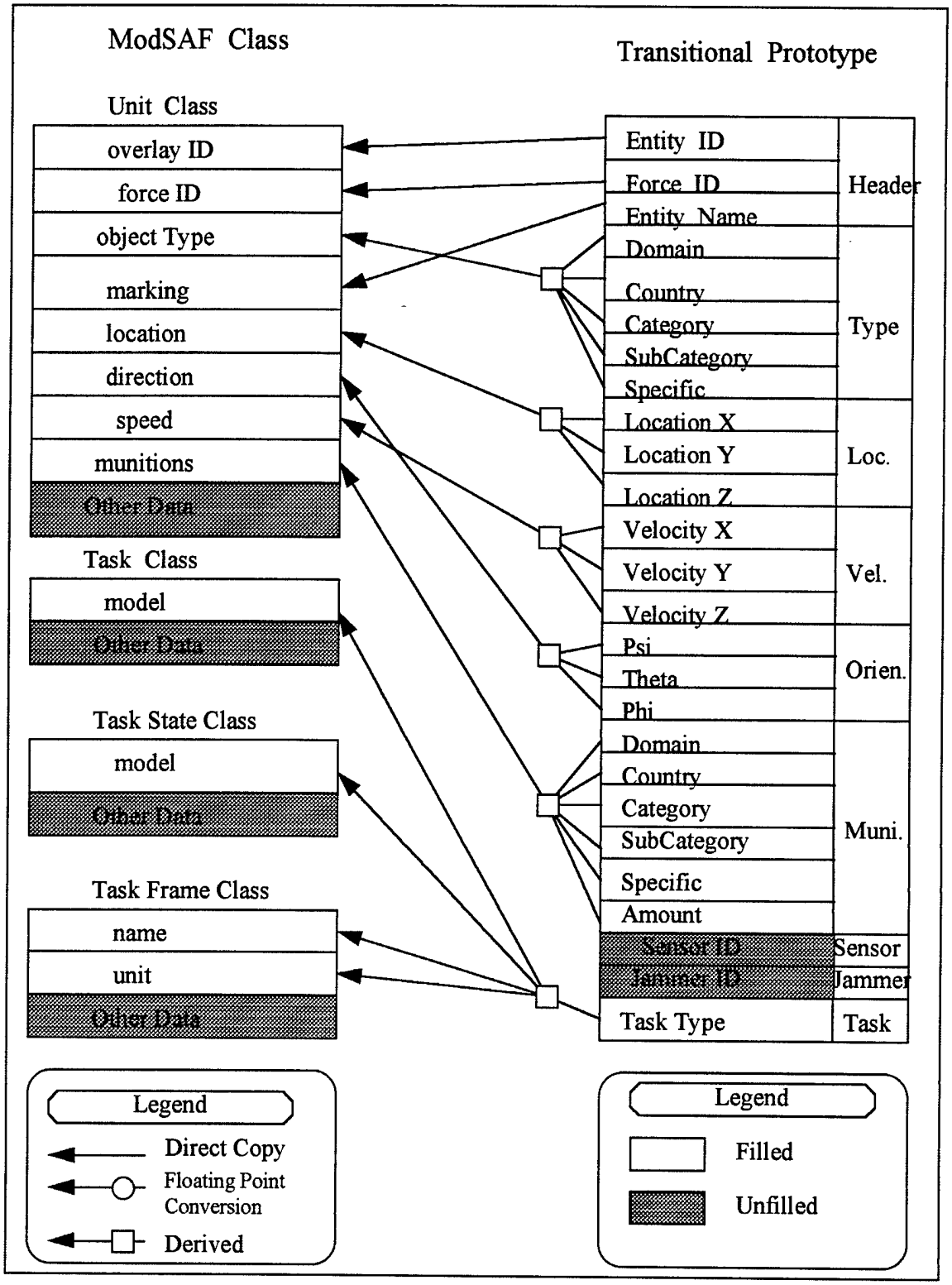

Figure 4.3 Remapping to a Transitional ModSAF Prototype

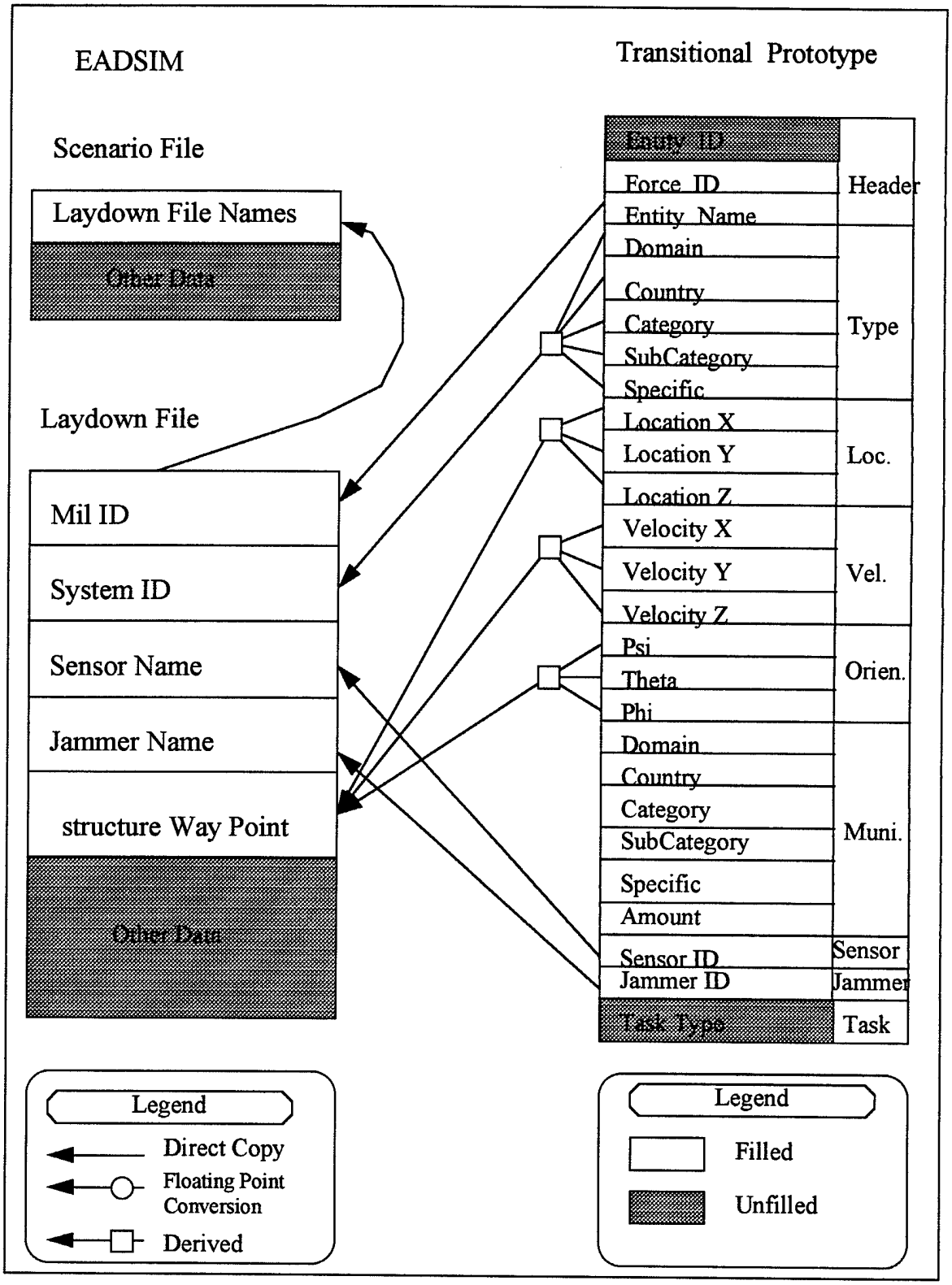

Figure 4.4 Remapping to a Transitional EADSIM Prototype

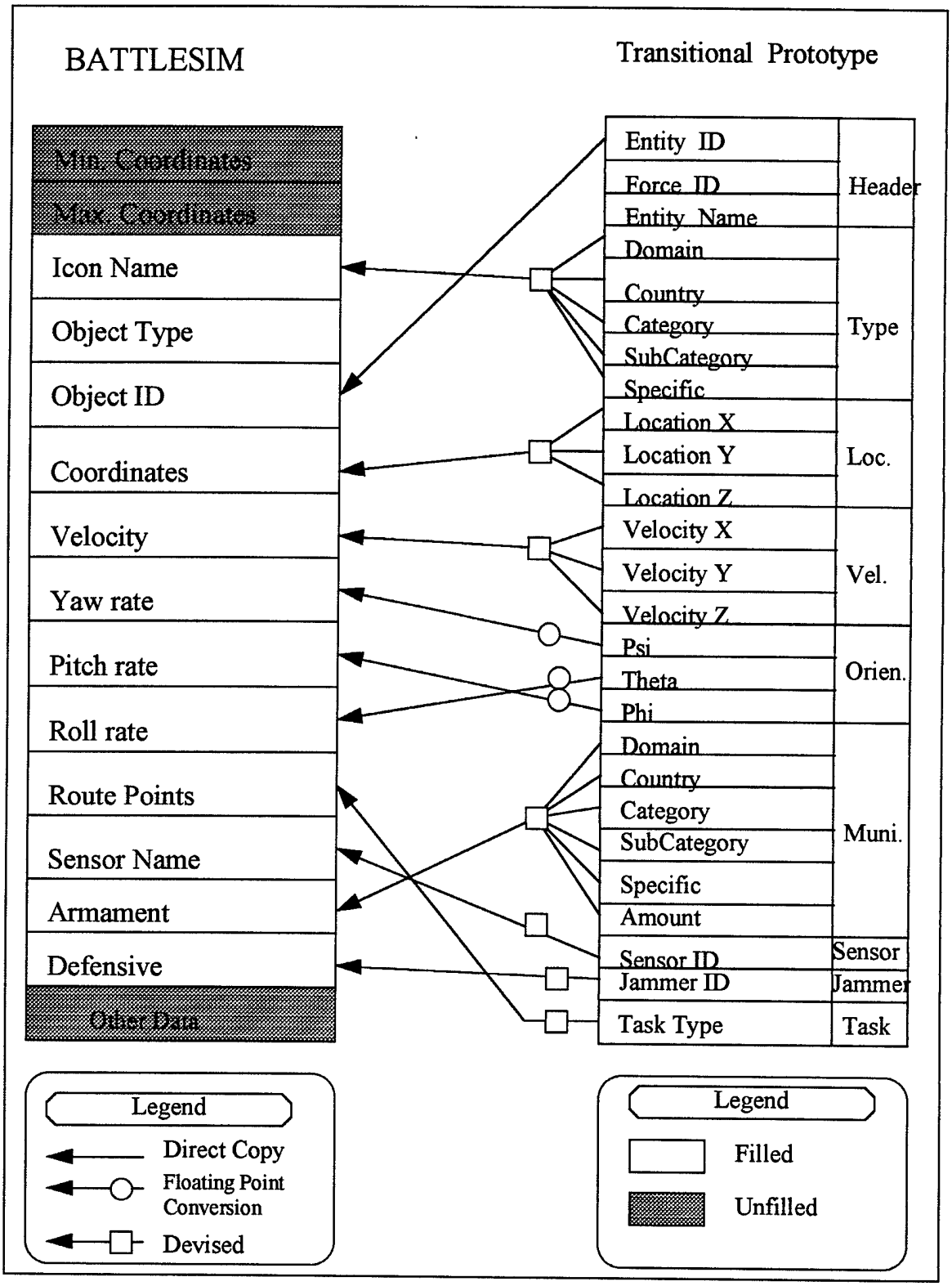

Figure 4.5 Remapping to a Transitional BATTLESIM Prototype

4.5 Saving a Transitional Prototype as a Target Scenario

The last phase of translating a source scenario to a target scenario is to write the remapped target prototype as a target scenario that a target simulation can read.

4.5.1 Saving as a ModSAF Scenario Since a PO_database is created in the process of mapping into the transitional ModSAF prototype, the task of saving the PO_database into an actual file is very simple. We use a function provided by the ModSAF program. When using the function, all the data in PO_database are stored into a user defined file name.

4.5.2 Saving as an EADSIM Scenario The first thing to do is to save a scenario file which an EADSIM can read from transitional EADSIM prototype. At this time, there is nothing to get from the transitional EADSIM prototype. Only the laydown file name is designated by user input and all information of a scenario is stored here. Several different file paths are stored in the EADSIM scenario file, and the directory path that is running the EADSIM is also needed. Because typing in the directory path whenever the Scenario File Translator is running is inefficient, we store the EADSIM running directory path in a files. This makes the SFT run more efficiently. The laydown file that stores the information of an entity is written to a text file fitting the transitional EADSIM prototype. The transitional EADSIM Laydown prototype can be saved in a straightforward manner since it is designed the same as the data structure of the laydown file.

4.5.3 Saving as a BATTLESIM Scenario To save a transitional BATTLESIM prototype data structure, we simply write the items in the structure to disk sequentially. Because there is no way to determine the terrain filename or the data about BATTLESIM section, thus, the user must provide the values when the file is saved.

4.5.4 Saving as a Transitional File Sometimes it is necessary to save the Transitional Prototype itself for further uses after mapping the source scenario to the TP. This is also efficient when testing a scenario in several other simulations since the SFT just needs to read a source scenario one time. Once the SFT reads a source scenario converts it, and saves it as a TP,

there is no need to re-read and remap the scenario for testing with other simulations. Just reading the Transitional File is enough to convert the source scenario to a target scenario. Saving as a Transitional File means to write the items that have been mapped to the TP in the format of Transitional Prototype. The SFT saves the TP to a text format. The user can easily understand the data since the SFT also writes some comments in the file.

4.6 User Interface

This section presents the result of the SFT user interface. The SFT provides two interfaces; GUI and Command Line Interface (CLI). Basically the CLI is provided to every machine even in the places where the GUI is also available.

4.6.1 Graphics User Interface (GUI) Figure 4.6 is a picture of the SFT main window. The row of menus at the top of the window (New, Run, Quit, and Help) allows immediate access to several functions. A selection of the 'New' button initializes the SFT. The 'Run' button causes the SFT to translate a source scenario to a user-defined target scenario according to the current settings. If a necessary value is omitted, the SFT displays an error message. The 'Quit' button terminates the program, and the 'Help' button displays program-specific information to the user.

The rest of the main window is called a scenario selection area. It is divided into two areas: the source scenario area and the target scenario area. Both sides have same purpose, namely, to select a scenario type and a scenario file name.

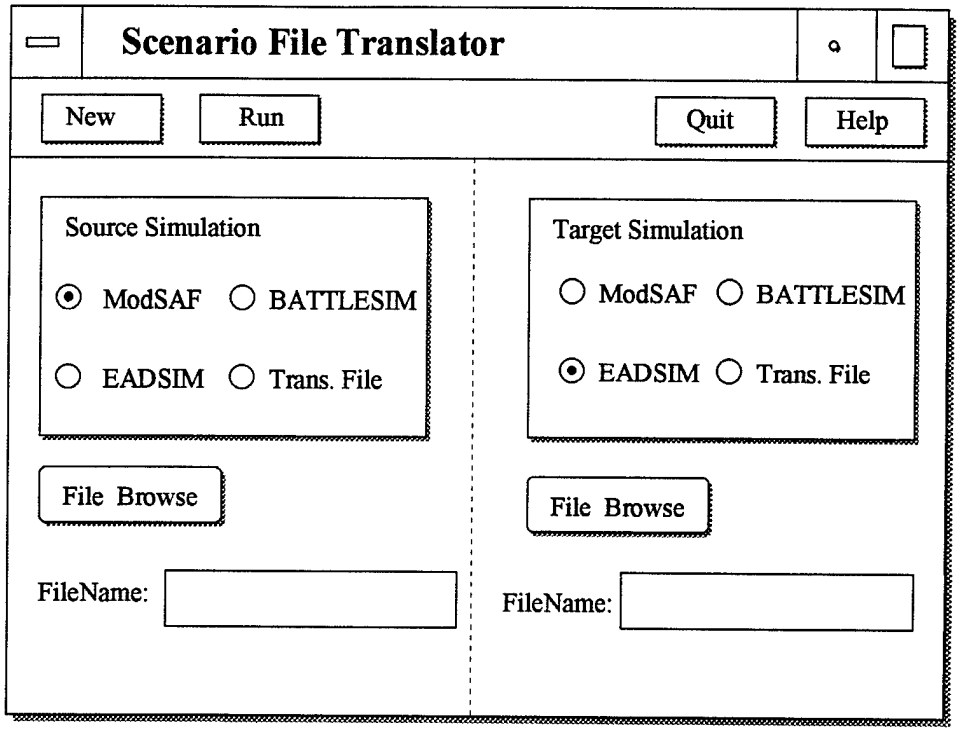

Figure 4.6 The Main Window of the SFT

The scenario selection area allows the user to choose the source/target simulation. The user can type a directory path and a file name in the 'FileName' display area or can use 'File Browse' button to search the directory structure for a file name. Error messages, warning messages, or processing messages are displayed in one of several different types of windows.

4.6.2 Command Line Interface A command line interface is also supported by the SFT for when the Motif library is not available. To use this interface, the user types the kind of a source/target simulation and the name of a source/target scenario file. The SFT then builds the target scenario file which the user defined. This interface has a validating function so it filters errant inputs. Appendix C shows the usage of SFT Command Line Interface.

4.7 Summary

This chapter described how the Scenario File Translator system is implemented. The SFT has five subsystems: the loading system, the mapping system, the remapping system, the saving system, and the user interface system. Actual data structures and methods for implementing the program followed the design specification. The next chapter presents the results of this system and offers suggestions for future work.

V. Results

5.1 Introduction

This chapter discusses the approach taken to ensure that the system, as implemented, fulfills the original requirements. Several cases are tested and the results are illustrated using a series of pictures taken within the SFT. This chapter also describes my observations of the test results.

5.2 The Scenario File Translator Test Cases

There are nine different test cases represented in the SFT. They are referred to as Test 1 through Test 9. Each test case represents a different type of conversion. This section shows what a source scenario looks like in a source simulation and not its translated scenario as represented in the target simulation.

5.2.1 Test 1 : ModSAF to EADSIM Figure 5.1 is a picture of a ModSAF simulation produced by ModSAF itself. It has 18 entities, and each entity has a different task. This scenario is used for Test 1, Test 2, and Test 3. Figure 5.2 shows the ModSAF scenario after EADSIM scenario and executed by EADSIM. It also has 18 entities which is the same as the number of entities in the ModSAF. Additionally, every entity location is the same as the one in the ModSAF, and every task for the ModSAF entities is also properly converted to the EADSIM scenario.

5.2.2 Test 2 : ModSAF to BATTLESIM Since the ModSAF scenario is already converted to the Translated File in the Test 1, there is no need to read the ModSAF scenario again. After converting the Transitional File to the BATTLESIM format, it also has 18 entities.

Once again, everything in the ModSAF scenario was properly translated to the BATTLESIM scenario.

5.2.3 Test 3 : ModSAF to ModSAF It is interesting to "convert" the ModSAF scenario to a ModSAF scenario after mapping it to the Transitional File, since the comparison between two scenarios demonstrates the fidelity of the conversion, showing clearly what is and is not mapped. Figure 5.3 shows a translated scenario from a ModSAF source scenario to a ModSAF target scenario. Notice, there are no changes between Figure 5.1 and Figure 5.3. The basic elements of the scenario are well translated with lost data during the mapping process restored by proper default values.

5.2.4 Test 4 : EADSIM to ModSAF Figure 5.4 is a snapshot of an EADSIM scenario. It has 10 entities, and each entity has a different flight path and mission. This particular EADSIM scenario is used for Test 4, Test 5, and Test 6. Figure 5.5 is a picture of a ModSAF simulation executing a converted scenario from the EADSIM by the SFT. Some entities which are not represented in ModSAF (such as an EADSIM 'Air Base') are translated into the 'unknown' entity.

5.2.5 Test 5 : EADSIM to BATTLESIM The number of entities in the BATTLESIM is the same as the number of entities in the EADSIM even after converting the scenario. Each BATTLESIM entity has the same route with an EADSIM entity route to perform the mission in the EADSIM source scenario.

5.2.6 Test 6 : EADSIM to EADSIM I also tested converting an EADSIM scenario to an EADSIM scenario through the Transitional File. Figure 5.6 shows the playback of the target scenario after its conversion by the SFT. There are no changes between Figure 5.4 and Figure 5.6 similar to that of Test 3, ModSAF to ModSAF. The result scenario file has just one laydown file. It successfully combined several laydown files into one laydown file.

5.2.7 Test 7 : BATTLESIM to ModSAF Figure 5.7 depicts ModSAF scenario as it executes the scenario after being converted from BATTLESIM by the SFT. BATTLESIM source scenario has 9 entities, and each entity has a different path. ModSAF shows each entity has its origin path and proper mission fitting the ModSAF task.

5.2.8 Test 8 : BATTLESIM to EADSIM Figure 5.8 shows a snapshot of a simulation run by EADSIM. There is the same number of entities in the BATTLESIM, and every entity location in the BATTLESIM is well translated into an EADSIM location. The entity path in BATTLESIM is also transferred into EADSIM, so every entity which has a path goes through with the translated path.

5.2.9 Test 9 : BATTLESIM to BATTLESIM I compared file format and file size after translating the original BATTLESIM scenario to the translated BATTLESIM scenario. Actually there are no differences between them. So, translating BATTLESIM scenario into another scenario has successfully worked.

5.4 Observation

We tested each possible case with our three simulations. The tests are demanding, because in each case the source scenario I produced most of its special purpose features. Thus, I could also test the interoperability between heterogeneous simulations.

According to these tests, we can make the following observations:

- Every entity type in a source scenario was completely translated into a target scenario.
- Every location in a source scenario is converted as the same position in a target scenario.
- The task definition in the TP covered every scenario's task type.
- The choice to use the DIS format data for Transitional Prototype (TP) such as entity type, entity location, entity velocity, entity orientation and entity munition type was a good decision.
- It was shown that using the intermediate format to correct a source scenario to a target scenario is a better method than a direct mapping. Assume there are currently N simulations. In case of a direct mapping method, $N*(N-1) = N^2 - N$ routines are needed; however, only $2*N$ routines are needed with using an intermediate format.

Based on these observations, the general objectives which are defined in Chapter Three are accomplished.

5.5 Summary

This chapter discussed the top-level of the SFT organization, and the test results. It also provided the observation of these test cases for converting a scenario to a target scenario. The next chapter formulates some conclusions and provides some recommendations for future work.

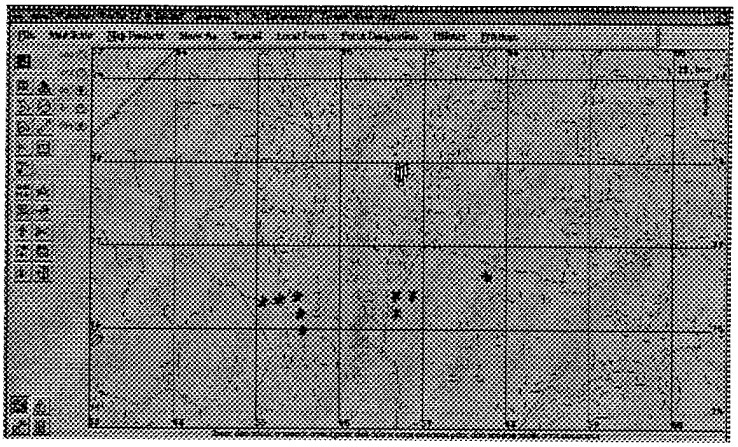

Figure 5.1 A Source Scenario of ModSAF

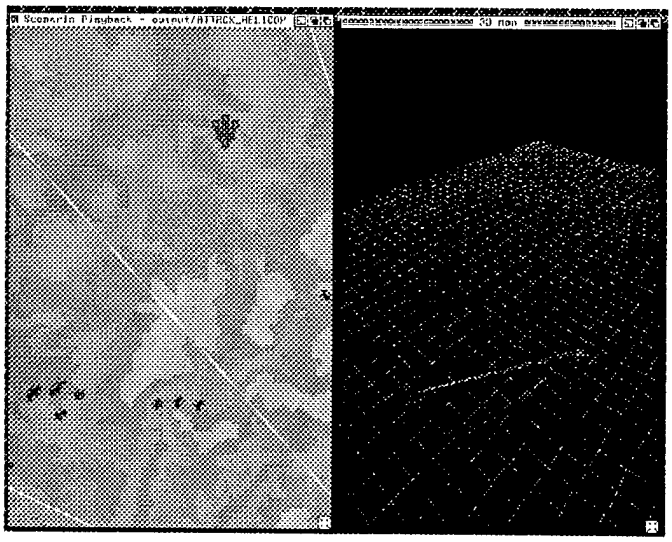

Figure 5.2 A Converted EADSIM Scenario from ModSAF

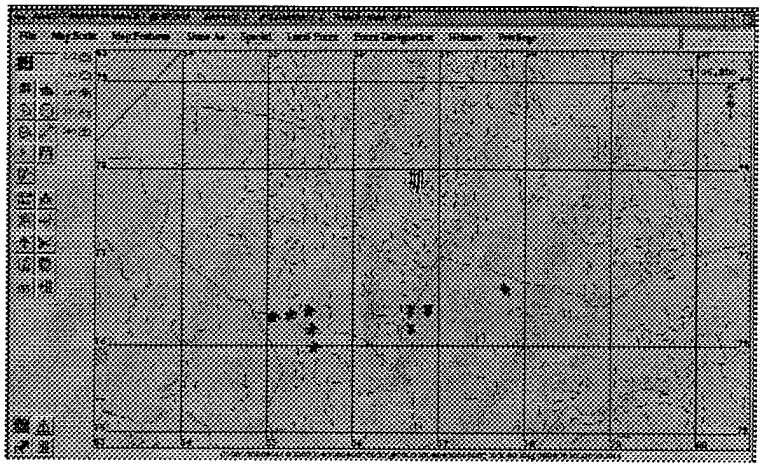

Figure 5.3 A Converted ModSAF Scenario from ModSAF

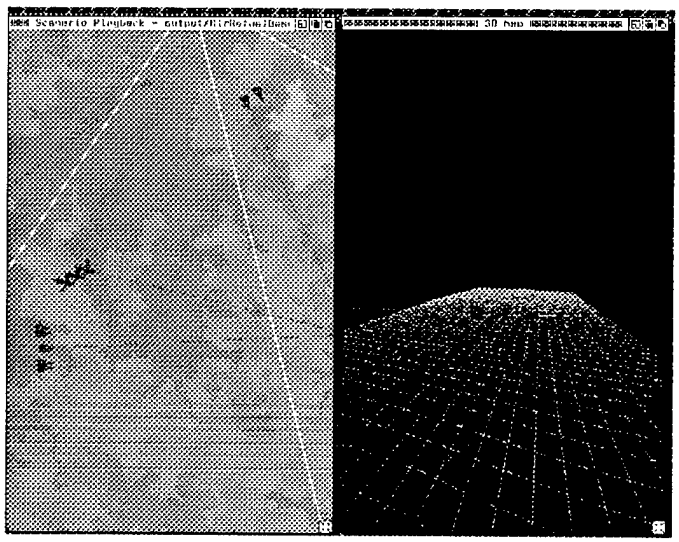

Figure 5.4 A Source Scenario of EADSIM

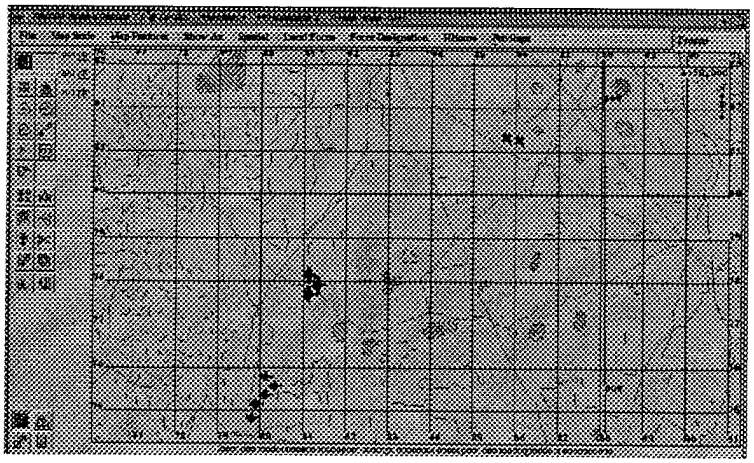

Figure 5.5 A Converted ModSAF Scenario from EADSIM

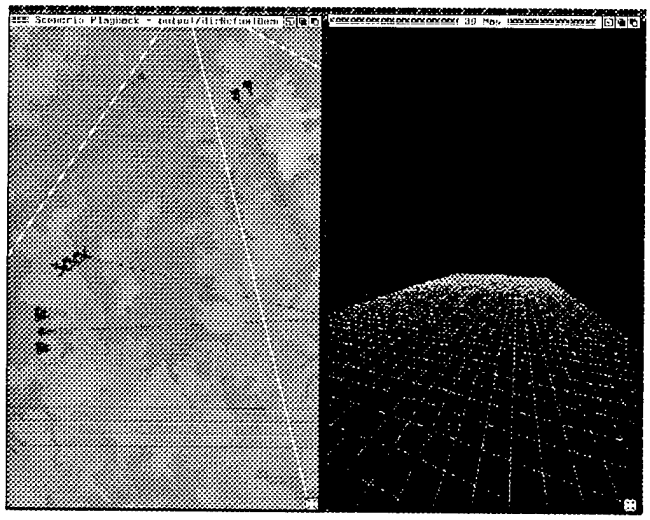

Figure 5.6 A Converted EADSIM Scenario from EADSIM

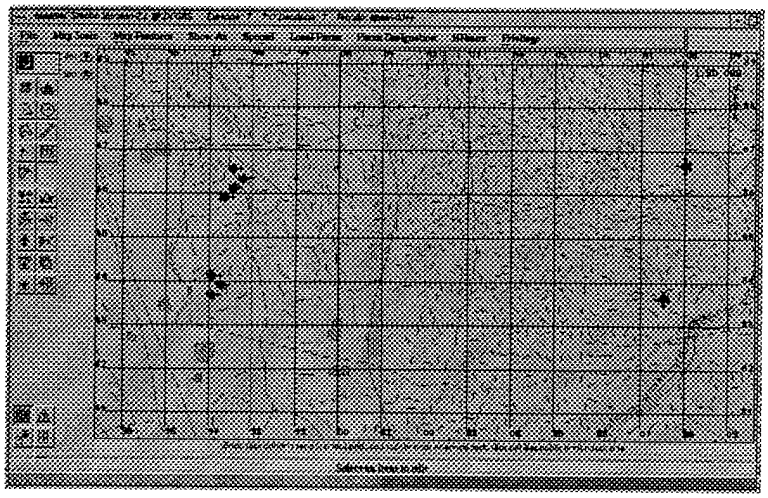

Figure 5.7 A Converted ModSAF Scenario from BATTLESIM

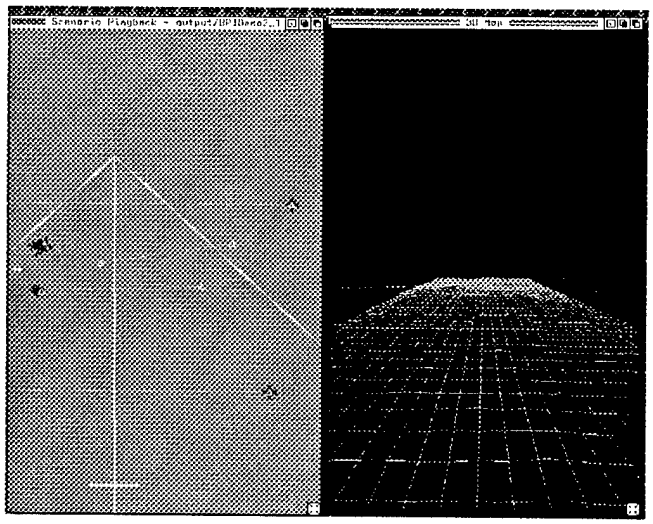

Figure 5.8 A Converted EADSIM Scenario from BATTLESIM

VI. Thesis Summary

6.1 Introduction

The effectiveness of computer simulation in the military is especially apparent, since real combat conditions are often costly to create for testing some scenarios. There now exists a variety of scenario generation and analysis products which have differing purposes and functions. Due to the particular purpose of each simulation, we cannot use one scenario of a simulation directly with another simulation without a translator since there is no standard scenario format. Interoperability between simulations is becoming more important in large scale simulation and distributed exercises. Translating scenarios to support interoperability is a useful process. I chose to remap to a target simulation after mapping into an intermediate prototype. This is more efficient because it is not dependent on the number of existing simulations. I proposed the transitional prototype as an intermediate format, and also developed the software tool (Scenario File Translator) that aids in the translation of scenario file formats with this transitional prototype so that scenarios developed by different simulations can be run by each other. After some test cases with the SFT, this scenario translating work has been achieved.

6.2 Recommendations for Future Work

There are several approaches for follow-on work in the SFT. One of the biggest areas of concern in the system is how the definition of the Transitional Prototype (TP) can be made more general. The prototype that covers all of the information of all simulation is the ideal format for this work. Table 6.1 is a snapshot of the work proposed in this chapter. Each of the primary system components – transitional prototype, DIS PDU, compatibility, and machine independence – is a candidate for future work.

6.2.1 Transitional Prototype As mentioned in Section 3.4.3, a set of information which covers all data for all simulations is the ideal format for the TP. However, this is not practical, and therefore we might focus on improving the data structures and the data formats in the TP as the best way to cover every simulations' scenario. As the TP data structure is generalized it will provide program for greater reusability and extensibility. A more detailed definition of the task type allows the target scenario to inherit the special purpose of a source scenario. We can also enhance the mapping process for more accurate translation. Since the more we map to the TP the more we extend the common area between simulations, even a small piece of data is important. Since the most widely-used default values are used when there is no represented data to a target scenario for mapping, finding a correct value for those data is another important process to increase the conversion percentage between two simulations.

6.2.2 DIS PDU Since most distributed simulations support DIS, grabbing the PDU from the network and building a state of the battlefield provides another advantage. Even if the mapping procedure of a simulation is not built, we can extract the current state of a scenario and can save it as the Transitional File which can be used by another simulation. The best PDU to determine the current state of a scenario is the 'Entity State PDU' (see Table 2.1). The PDU has information of each entity such as entity type, velocity, and location. Using the PDU we can rebuild a scenario which contains all the simulations on the network. Of course, we can select some specific PDUs by 'Exercise ID' in PDU Header. However, since DIS PDUs send current events over the network, it is not possible to determine the task of an

Table 6.1 Proposed Future Work

Functional Area	Proposed Work
Transitional Prototype	Improving data structure and data format
	Extend the type of task
	Enhance accuracy of mapping process
	Correct inserting for default values
DIS PDU	Develop DIS "Entity State PDU" snapshot function
	Select specific PDU with Exercise ID in PDU Header
	Changing TP to DIS PDU
	Send converted DIS PDU from TP via network
	Receive other DIS PDUs relating the TP
Machine Independence	Provide 'Makefile' to compile on any machine
	Analyze the computer system to allocate optimized memory size
	Enhance stability and extensibility
Compatibility	Accommodate other prototype
	Transfigure to other format
	Accompany with DIS PDU

entity as described in Section 3.2.1.1. As DIS PDU is more well developed we can alleviate this problem. Conversely, we can build a procedure which changes a TP to PDUs because the TP already uses the DIS format. That is another benefit since we can send the TP via the network through the DIS PDU, providing for program extensibility.

6.2.3 Machine Independence It is desirable for the SFT to run on any machine to support many currently-existing simulations. Of course, there is a way to translate a scenario on one type of machine to a target scenario on another type of machine. When the scenarios are transferred to the machine which has the SFT, a target scenario can be made and can be sent to the machine which uses the target scenario. However, it depends on the network state. For more stable support, it is necessary to compile on any machine. The flexible and comprehensive 'Makefile' supports this.

6.2.4 Compatibility I chose DIS format to save some scenario data such as entity type, entity location, and munition type. If a more well-defined format to send a current state of a scenario is developed, or if there is another format to describe data of scenarios, compatibility with these formats extends the usability of the SFT.

6.2.5 Memory Size In the current program, all data of the source/target scenario and TP are stored in computer memory. The required memory size is described in Section 4.2. The potential problem is an out-of-memory exception. Memory size is not a big problem in this area since we can increase computer memory relatively inexpensively. However, we need to consider the possibility that the memory is not sufficient to store the whole data of a scenario which has a hundred thousand entities, or a million entities, or a billion entities in extreme cases. Of course, there are several techniques to solve this problem. The more we minimize the required memory size, the more the SFT is stable. One of the ways to minimize the required memory size is to optimize the TP data structure. When we determine the best size of each variable, we can reduce the TP memory size. When the memory is not enough even after optimizing the data structure, another way is to use a hard disk drive to temporarily save current state of the TP.

6.3 Conclusion

This thesis effort produces a system which translates scenarios of one format to scenarios of a different format in order to increase interoperability between simulations. Although this thesis effort concentrated on three specific simulations ModSAF, EADSIM and BATTLESIM, this work can also be applied to other simulations, and that task is simplified by the TP.

Appendix A. Transitional Scenario File Prototype

A.1 ModSAF

Table A.1 ModSAF Transitional Prototype

Type	Variable Name	Remark
PO_FILE_HDR *	ms_hdr	structure pointer
PO_FILE_ENTRY	ms_fe	array size : [0xFFFFF]
void *	ms_class	address pointer

PO_FILE_HDR, PO_FILE_ENTRY : refer to ModSAF source code 'p_po.h'

A.2 EADSIM

A.2.1 EADSIM Scenario File

Table A.2 EADSIM 'Scenario File' Transitional Prototype

Type	Variable Name	Remark
es_header *	ES_HeaderAddr	structure pointer
es_laydownfile *	ES_LaydownFileAddr	structure pointer
es_elementmap *	ES_ElementMapAddr	structure pointer
es_networkfile *	ES_NetworkfileAddr	structure pointer
es_zerothrumaxhost *	ES_ZeroThruMaxHostAddr	structure pointer
es_unusedstring *	ES_UnusedStringAddr	structure pointer
es_degug *	ES_DebugAddr	structure pointer
es_host *	ES_HostAddr	structure pointer
es_maxfilec3ioutput *	ES_MaxFileC3IOutputAddr	structure pointer
es_truth *	ES_TruthAddr	structure pointer
es_spds *	ES_SPDSAddr	structure pointer
es_log *	ES_LogAddr	structure pointer
es_stat *	ES_StatAddr	structure pointer
es_pstat *	ES_PStatAddr	structure pointer
es_file *	ES_FileAddr	structure pointer
es_earthradadj *	ES_EarthRadAdjAddr	structure pointer
es_hzululocal *	ES_HZuluLocalAddr	structure pointer
es_displaythruroute *	ES_DisplayThruRouteAddr	structure pointer
es_formatthrustarting *	ES_FormatThruStartingAddr	structure pointer
es_process *	ES_ProcessAddr	structure pointer
es_write *	ES_WriteAddr	structure pointer
es_c3ilog *	ES_C3ILogAddr	structure pointer
es_image *	ES_ImageAddr	structure pointer
es_transradfilepath *	ES_TransRadFilePathAddr	structure pointer
es_time *	ES_TimeAddr	structure pointer
es_external *	ES_ExternalAddr	structure pointer
es_tb *	ES_TBAddr	structure pointer

Table A.3 EADSIM 'es_filename' data structure

Type	Variable Name	Remark
char	FileName	string size : 79
es_filename *	NextFileNameAddr	structure pointer

Table A.4 EADSIM 'es_header' data structure

Type	Variable Name	Remark
char	Version	string size : 79
char	Title	string size : 79
int	StartTime	
int	EndTime	
int	Interval	
unsigned long int	MaxDuration	

Table A.5 EADSIM 'es_laydownfile' data structure

Type	Variable Name	Remark
int	NumLaydowns	
es_filename *	NextFileNameAddr	structure pointer

Table A.6 EADSIM 'es_elementmap' data structure

Type	Variable Name	Remark
char	ElementPath	string size : 79
char	MapPath	string size : 79

Table A.7 EADSIM 'es_networkfile' data structure

Type	Variable Name	Remark
int	NumNetworkFile	string size : 79
es_filename *	NextFileNameAddr	structure pointer

Table A.8 EADSIM 'es_zerothrumaxhost' data structure

Type	Variable Name	Remark
unsigned char	Zero	value : 0
char	AOIPath	stringsize : 79
unsigned char	HOSTILE_SEE_HOSTILE	boolean
unsigned char	FRIENDLY_SEE_FRIENDLY	boolean
unsigned char	MAX_HOST	value : 8

Table A.9 EADSIM 'es_unusedstring' data structure

Type	Variable Name	Remark
char	UnusedString0	string size : 79
char	UnusedString1	string size : 79
char	UnusedString2	string size : 79
char	UnusedString3	string size : 79
char	UnusedString4	string size : 79
char	UnusedString5	string size : 79
char	UnusedString6	string size : 79
char	UnusedString7	string size : 79

Table A.10 EADSIM 'es_debug' data structure

Type	Variable Name	Remark
int	Debug_FP	
int	Debug_C3I	
int	Debug_DET	
int	Debug_PROP	
int	Debug_GRF	
int	Debug_NAM	
int	Debug_ALSP	
int	Debug_TB	

Table A.11 EADSIM 'es_host' data structure

Type	Variable Name	Remark
char	Host_FP	string size : 79
char	Host_C3I	string size : 79
char	Host_DET	string size : 79
char	Host_PROP	string size : 79
char	Host_GRF	string size : 79
char	Host_NAM	string size : 79
char	Host_ALSP	string size : 79
char	Host_TB	string size : 79

Table A.12 EADSIM 'es_maxfilec3ioutput' data structure

Type	Variable Name	Remark
unsigned char	MAX_FILE	value : 40
char	C3IOutputPath	string size : 79

Table A.13 EADSIM 'es_truth' data structure

Type	Variable Name	Remark
char	C3ITruthFile	string size : 79
char	FPTruthFile	string size : 79
char	PropagationTruthFile	string size : 79
char	DetectionTruthFile	string size : 79

Table A.14 EADSIM 'es_spds' data structure

Type	Variable Name	Remark
char	C3ISPDSFile	string size : 79
char	DetectionSPDSFile	string size : 79

Table A.15 EADSIM 'es_log' data structure

Type	Variable Name	Remark
char	C3ILOGFile	string size : 79
char	FPLOGFile	string size : 79
char	PropagationLOGFile	string size : 79
char	DetectionLOGFile	string size : 79

Table A.16 EADSIM 'es_stat' data structure

Type	Variable Name	Remark
char	C3ISTATFile	string size : 79
char	FPSTATFile	string size : 79
char	PropagationSTATFile	string size : 79

Table A.17 EADSIM 'es_pstat' data structure

Type	Variable Name	Remark
char	C3IPSTATFile	string size : 79
char	DetectionPSTATFile	string size : 79
char	FPPSTATFile	string size : 79
char	PropagationPSTATFile	string size : 79

Table A.18 EADSIM 'es_file' data structure

Type	Variable Name	Remark
char	TIMESLOTFile	string size : 79
char	MSGDATAFile	string size : 79
char	C3IENGSTATFile	string size : 79
char	C3ICOMMSTATFile	string size : 79
char	C3ITRACKSTATFile	string size : 79
char	FPSTATHEADERFile	string size : 79
char	COMMREQFile	string size : 79
char	CONSTATFile	string size : 79
char	ASSETFile	string size : 79
char	DetectionSTATFile	string size : 79
char	DetectionDEBUGFile	string size : 79
char	MEZFile	string size : 79
char	FPOutputPath	string size : 79
char	DetectionOutputPath	string size : 79
char	PropagationOutputPath	string size : 79
char	C3IINTELSTATFile	string size : 79
char	C3IBASESTATFile	string size : 79
char	C3IIFFSTATFile	string size : 79
char	FPEVENTFile	string size : 79
char	C3IJAMSTATFile	string size : 79
char	PREFLYTBMFile	string size : 79
char	C3IRSRCSTATFile	string size : 79

Table A.19 EADSIM 'es_earthradadj' data structure

Type	Variable Name	Remark
float	RadarEarthRadAdj	
float	SigIntEarthRadAdj	
float	HumIntEarthRadAdj	
float	ImintEarthRadAdj	
float	IRLEarthRadAdj	
float	IREEarthRadAdj	

Table A.20 EADSIM 'es_hzululocal' data structure

Type	Variable Name	Remark
int	HStart	
int	Zulustart	
int	LocalStart	

Table A.21 EADSIM 'es_displaythruroute' data structure

Type	Variable Name	Remark
char	DisplayPreFile	string size : 79
char	ColorPreFile	string size : 79
char	LLTRFile	string size : 79
char	RouteFile	string size : 79

Table A.22 EADSIM 'es_formatthrustarting' data structure

Type	Variable Name	Remark
int	FormatSPDS	
int	JammerOnTime	Not used
int	FractionPropUpdate	
int	Socket_MAXSOCKET	
int	TotalRuns	
int	RunsToMake	
int	StartingRun	

Table A.23 EADSIM 'es_process' data structure

Type	Variable Name	Remark
unsigned char	FP_PROCESS	boolean
unsigned char	C3I_PROCESS	boolean
unsigned char	DET_PROCESS	boolean
unsigned char	PROP_PROCESS	boolean
unsigned char	SGEN_PROCESS	boolean
unsigned char	NAM_PROCESS	boolean
unsigned char	ALSP_PROCESS	boolean
unsigned char	DIS_PROCESS	boolean
unsigned char	TB_PROCESS	boolean
unsigned char	SEEDFROMPREVIOUS	boolean

Table A.24 EADSIM 'es_write' data structure

Type	Variable Name	Remark
unsigned char	WRITE_PROP_LOG	boolean
unsigned char	WRITE_PROP_STAT	boolean
unsigned char	WRITE_PROP_PSTAT	boolean
unsigned char	WRITE_DET_SPDS	boolean
unsigned char	WRITE_LOG	boolean
unsigned char	WRITE_DET_STAT	boolean
unsigned char	WRITE_DET_PSTAT	boolean
unsigned char	WRITE_DET_DEBUG	boolean
unsigned char	WRITE_C3I_LOG	boolean
unsigned char	WRITE_C3I_PSTAT	boolean
unsigned char	WRITE_ENGSTAT	boolean
unsigned char	WRITE_IFFSTAT	boolean
unsigned char	WRITE_COMMSTAT	boolean
unsigned char	WRITE_TRACKSTAT	boolean
unsigned char	WRITE_INTELSTAT	boolean
unsigned char	WRITE_BASESTAT	boolean
unsigned char	WRITE_JAMSTAT	boolean
unsigned char	WRITE_PSRCSTAT	boolean
unsigned char	WRITE_TRUTH	boolean : TRUE
unsigned char	WRITE_FP_LOG	boolean
unsigned char	WRITE_FP_PSTAT	boolean
unsigned char	WRITE_STATHDR	boolean : TRUE

Table A.25 EADSIM 'es_c3ilog' data structure

Type	Variable Name	Remark
unsigned char	WRITE_FPEVENTS	boolean
unsigned char	C3ILog_PLAT_STATUS	boolean
unsigned char	C3ILog_MSG_CONFIRM	boolean
unsigned char	C3ILog_PHASE_STATS	boolean
unsigned char	C3ILog_UTIL_STATS	boolean
unsigned char	C3ILog_MSG_HANDLER	boolean
unsigned char	C3ILog_TRACK_HANDL	boolean
unsigned char	C3ILog_ENG_EVENTS	boolean
unsigned char	C3ILog_SPDS_INFO	boolean
unsigned char	C3ILog_MISL_DETECT	boolean
unsigned char	C3ILog_COMM_CHECKS	boolean
unsigned char	C3ILog_IFF_MSGS	boolean
unsigned char	C3ILog_RSRC_EVENTS	boolean
unsigned char	C3ILog_SENSOR_MAN	boolean
unsigned char	LOG_ALL_NET_STATS	boolean
float	NetStatsInterval	boolean

Table A.26 EADSIM 'es_image' data structure

Type	Variable Name	Remark
unsigned char	DTED_IMAGE	
unsigned char	ADRG_IMAGE	
unsigned char	NO_IMAGE	
unsigned char	ADRGMapImageType	1: Overview 3 : Map Image
float	ADRGMapNLat	
float	ADRGMapSLat	
float	ADRGMapWLon	
float	ADRGMapELon	
unsigned char	ADRG_USE_CDROM_ASPECT	
unsigned char	ADRGImageSource	1:Read from file 2:from CDROM
char	ADRGImagePath	string size : 79

Table A.27 EADSIM 'es_transradfilepath' data structure

Type	Variable Name	Remark
char	TransmissionFilePath	string size : 79
char	RadianceFilePath	string size : 79

Table A.28 EADSIM 'es_defaultsentime' data structure

Type	Variable Name	Remark
int	DefaultSenOnTime	
int	DefaultSenOffTime	
es_defaultsentime *	ES_NextDefaultSenTimeAddr	structure pointer

Table A.29 EADSIM 'es_defaultjamtime' data structure

Type	Variable Name	Remark
int	DefaultJamOnTime	
int	DefaultJamOffTime	
es_defaultjamtime *	ES_NextDefaultJamTimeAddr	structure pointer

Table A.30 EADSIM 'es_totalruns' data structure

Type	Variable Name	Remark
unsigned int	RSeedC3I	
unsigned int	RSeedDetect	
es_totalruns *	ES_NextTotalRunsAddr	structure pointer

Table A.31 EADSIM 'es_time' data structure

Type	Variable Name	Remark
unsigned int	NumGlobalSensorTime	
es_defaultsentime *	ES_NextDefaultSenTimeAddr	structure pointer
unsigned int	NumGlobalJammerTimes	
es_defaultjamtime *	ES_NextDefaultJamTimeAddr	structure pointer
es_totalruns *	ES_NextTotalRunsAddr	structure pointer

Table A.32 EADSIM 'es_external' data structure

Type	Variable Name	Remark
unsigned int	ALSPAddress_ConfedID	
unsigned int	ALSPAddress_ActorID	
char	ALSPAddress_ACMHostName	string size : 79
unsigned int	DISAddress_Site	
unsigned int	DISAddress_Host	
int	BCastInPort	
int	BCastOutPort	
char	NetInterface	string size : 25
int	ExerciseID	
float	AirTimeOut	
float	GndTimeOut	
double	AirThreshold	
double	GndThreshold	
unsigned char	MSL_LAUNCH_PROCESS	

Table A.33 EADSIM 'es_tb' data structure

Type	Variable Name	Remark
char	TBAddress_HostMachine	string size : 25
char	TBAddress_ExePath	string size : 254
char	TBAddress_OutPath	string size : 254
char	TBAddress_DataPath	string size : 254
char	TBAddress_CmdLine	string size : 1024
unsigned int	TBAddress_Site	
unsigned int	TBAddress_Host	
unsigned char	EXT_REAL_TIME	
unsigned char	TIME_REGULATING	
unsigned char	TIME_CONSTRAINED	
unsigned char	START_TB	
unsigned char	PREFLYTBM	
float	EMCONLat	
float	EMCONLon	

A.2.2 EADSIM Laydown File

Table A.34 EADSIM 'es_lay' data structure

Type	Variable Name	Remark
char	Version	string size : 25
int	NumPlatforms	
es_layplatform *	ES_NextLayPlatformAddr	structure pointer

Table A.35 EADSIM 'es_layplatform' data structure

Type	Variable Name	Remark
char	MilID	string size : 25
char	SystemID	string size : 25
char	CommanderID	string size : 25
int	ColorIndex	
es_laysensor *	ES_NextLaySensorAddr	structure pointer
int	NumComDevs	
es_laycomdev *	ES_NextLayComDevAddr	structure pointer
es_layjammer *	ES_NextLayJammerAddr	structure pointer
int	NumAssets	
es_layasset *	ES_NextLayAssetAddr	structure pointer
int	NumTargets	
es_laytarget *	ES_NextLayTargetAddr	structure pointer
int	NumIFTUPlatforms	
es_layiftuplatform *	ES_NextLayIFTUPlatformAddr	structure pointer
unsigned char	SIM	boolean
unsigned char	HHOUR	boolean
unsigned char	LOCAL	boolean
unsigned char	ZULU	boolean
unsigned char	HM_NO_DELIM	boolean
unsigned char	HMS	boolean
unsigned char	AS_ENTERED	boolean
char	TimeSuffix	string size : 25
unsigned char	LOG_TRACK_EVENT	
unsigned char	LASER_ABSOLUTE	
float	LaserRestAz	
float	LaserRestEl	
unsigned char	PlatformType	
es_layplatformsatellite *	ES_LayPlatformSatelliteAddr	structure pointer
unsigned char	UseRouteData	boolean
us_layuseroutedata *	ES_LayUseRouteDataAddr	structure pointer
us_laynotuseroutedata *	ES_LayNotUseRouteDataAddr	structure pointer
int	NumPTLs	
es_layptl *	ES_NextLayPTLAddr	structure pointer
int	NumLCS	
es_laymilid *	ES_NextLayLCSAddr	structure pointer
int	NumDecoys	
es_laymilid *	ES_NextLayDecoyAddr	structure pointer
int	NumSurvPlat	
es_laymilid *	ES_NextLaySurvAddr	structure pointer
unsigned char	CMData	boolean
unsigned char	EMCOM_DEFAULT	boolean
int	EMCOMAuthority	
es_layplatformaircraft *	ES_LayPlatformAircraft	
unsigned char	AIRBASE	boolean
es_layplatformairbase *	ES_LayPlatformAirBaseAddr	structure pointer
es_layplatform *	ES_NextLayPlatformAddr	structure pointer

Table A.36 EADSIM 'es_laysensor' data structure

Type	Variable Name	Remark
int	NumSensors	
es_laysensorheader *	ES_NextlaySensorHeaderAddr	structure pointer
int	NumTimings	
int	NumTimes	
es_laytime *	ES_NextlayTimeAddr	structure pointer

Table A.37 EADSIM 'es_laysensorheader' data structure

Type	Variable Name	Remark
char	Name	string size : 25
int	AntHeight	
float	VerAngle	
float	HorAngle	
float	Lat	
float	Lon	
float	Alt	
unsigned char	PtMode	
float	NomFreq	
char	ToPlatform	string size : 25
unsigned char	DEFAULT_ANTHEIGHT	boolean
unsigned char	DEFAULT_HORANGLE	boolean
unsigned char	DEFAULT_VERANGLE	boolean
unsigned char	DEFAULT_PTMODE	boolean
unsigned char	DEFAULT_TIMING	boolean
unsigned char	DEFAULT_FREQ	boolean
int	NumSensorPointings	
es_laypointing *	ES_NextLayPointingAddr	structure pointer
es_laysensorheader *	ES_NextLaySensorHeaderAddr	structure pointer

Table A.38 EADSIM 'es_laycomdev' data structure

Type	Variable Name	Remark
char	Name	string size : 25
int	AntHeight	
float	VerAngle	
float	HorAngle	
float	Lat	
float	Lon	
float	Alt	
unsigned char	PtMode	
char	ToPlatform	string size : 25
unsigned char	DEFAULT_ANTHEIGHT	boolean
unsigned char	DEFAULT_HORANGLE	boolean
unsigned char	DEFAULT_VERANGLE	boolean
unsigned char	DEFAULT_PTMODE	boolean
es_laycomdev *	ES_NextLayComDevAddr	structure pointer

Table A.39 EADSIM 'es_layjammer' data structure

Type	Variable Name	Remark
int	NumJammers	
es_layjammerheader *	ES_NextlayJammerHeaderAddr	structure pointer
int	NumTimings	
int	NumTimes	
es_laytime *	ES_NextlayTimeAddr	structure pointer

Table A.40 EADSIM 'es_layjammerheader' data structure

Type	Variable Name	Remark
char	Name	string size : 25
int	Number	Not used
int	AntHeight	
float	VerAngle	
float	HorAngle	
float	Lat	
float	Lon	
float	Alt	
unsigned char	PtMode	
char	ToPlatform	string size : 25
unsigned char	DEFAULT_ANTHEIGHT	boolean
unsigned char	DEFAULT_HORANGLE	boolean
unsigned char	DEFAULT_VERANGLE	boolean
unsigned char	DEFAULT_PTMODE	boolean
unsigned char	DEFAULT_TIMING	boolean
int	NumJammerPointings	
es_laypointing *	ES_NextLayPointingAddr	structure pointer
es_layjammerheader *	ES_NextLayJammerHeaderAddr	structure pointer

Table A.41 EADSIM 'es_layasset' data structure

Type	Variable Name	Remark
char	Name	string size : 25
int	Priority	
float	ThreatRangeCEN	
float	CriticalRangeCEN	
float	ThreatRangeDECEN	
float	CriticalRangeDECEN	
float	DownRangeCEN	
float	CrossRangeCEN	
float	DownRangeDECEN	
float	CrossRangeDECEN	
unsigned char	CEN_ABT_SELF	boolean
unsigned char	CEN_ABT_SUBORD	boolean
unsigned char	CEN_ABT_CMDR	boolean
unsigned char	CEN_ABT_ASSET	boolean
unsigned char	CEN_ABT_SUBSUBORD	boolean
unsigned char	CEN_ABT_SUBASSET	boolean
unsigned char	CEN_ABT_ZONE	boolean
unsigned char	DECEN_ABT_SELF	boolean
unsigned char	DECEN_ABT_SUBORD	boolean
unsigned char	DECEN_ABT_CMDR	boolean
unsigned char	DECEN_ABT_ASSET	boolean
unsigned char	DECEN_ABT_SUBSUBORD	boolean
unsigned char	DECEN_ABT_SUBASSET	boolean
unsigned char	DECEN_ABT_ZONE	boolean
unsigned char	DEFEND_TM	boolean
unsigned char	CEN_TM_SELF	boolean
unsigned char	CEN_TM_SUBORD	boolean
unsigned char	CEN_TM_CMDR	boolean
unsigned char	CEN_TM_ASSET	boolean
unsigned char	CEN_TM_SUBSUBORD	boolean
unsigned char	CEN_TM_SUBASSET	boolean
unsigned char	CEN_TM_ZONE	boolean
unsigned char	DECEN_TM_SELF	boolean
unsigned char	DECEN_TM_SUBORD	boolean
unsigned char	DECEN_TM_CMDR	boolean
unsigned char	DECEN_TM_ASSET	boolean
unsigned char	DECEN_TM_SUBSUBORD	boolean
unsigned char	DECEN_TM_SUBASSET	boolean
unsigned char	DECEN_TM_ZONE	boolean
unsigned char	DEFEND_ABT	boolean
es_layasset *	ES_NextLayAssetAddr	structure pointer

Table A.42 EADSIM 'es_laytarget' data structure

Type	Variable Name	Remark
char	Name	string size : 25
int	Priority	
int	LaunchTime	
int	WPNumber	
char	Weapon	string size : 25
char	CaptivePlatform	string size : 25
float	LauchDelay	
unsigned char	TARGET_REQDETECT	boolean
float	HOB	
es_laytarget *	ES_NextLayTargetAddr	structure pointer

Table A.43 EADSIM 'es_layiftuplatform' data structure

Type	Variable Name	Remark
char	Name	string size : 25
es_layiftuplatform *	ES_NextLayIFTUPlatformAddr	structure pointer

Table A.44 EADSIM 'es_layplatformsatellite' data structure

Type	Variable Name	Remark
float	X	
float	Y	
float	Z	
float	XDot	
float	YDot	
float	ZDot	
float	InitialTime	
float	InitialLon	

Table A.45 EADSIM 'es_layuseroutedata' data structure

Type	Variable Name	Remark
char	RouteName	string size : 25
float	ActivationTime	
float	ActivationTimeDt	

Table A.46 EADSIM 'es_laynotuseroutedata' data structure

Type	Variable Name	Remark
int	NumWayPoints	
unsigned char	WpMode	
es_laywaypoint *	ES_NextLayNURDWayPointAddr	structure pointer

Table A.47 EADSIM 'es_layptl' data structure

Type	Variable Name	Remark
int	PTLOnTime	
float	PTLAzimuth	
es_layptl *	ES_NextLayPTLAddr	structure pointer

Table A.48 EADSIM 'es_layacfrc' data structure

Type	Variable Name	Remark
float	RefuelAmount	
int	NumTankers	
es_laymilid *	ES_NextLayMilIDAddr	structure pointer

Table A.49 EADSIM 'es_laywaypoint' data structure

Type	Variable Name	Remark
float	Lat	
float	Lon	
float	Alt	
unsigned char	Type	
char	MilID	string size : 25
unsigned char	TerrainFlag	
unsigned char	Zmode	
float	Speed	
int	OnTime	
int	OffTime	
es_laywaypoint *	ES_NextLayWayPointAddr	structure pointer

Table A.50 EADSIM 'es_layart' data structure

Type	Variable Name	Remark
int	NumWayPoints	
es_laywaypoint *	ES_NextLayWaypointAddr	structure pointer

Table A.51 EADSIM 'es_layplatformaircraft' data structure

Type	Variable Name	Remark
char	FlightLeader	string size : 25
char	HomeAirbaseID	string size : 25
int	AlertLevel	
unsigned char	AtBaseFlag	
unsigned char	FLIGHT_REFILL	boolean
int	DeactivationTime	
unsigned char	FLIGHT_REFUEL_CAPABLE	boolean
es_layacfrc *	ES_LayACFRCAddr	structure pointer
unsigned char	AIRREFUELTANKER	boolean
es_layart *	ES_LayACARTAddr	structure pointer

Table A.52 EADSIM 'es_layplatformairbase' data structure

Type	Variable Name	Remark
float	Radius	
float	RunWayArea	
float	TurnArea	
float	C2Area	
float	AC_PK	
float	TOIntervalPenalty	
int	TurnDelay	
int	DelayFirstImpact	
int	MinTakeOffInterval	
int	TOPenaltyExp	
int	TBMpriority	
int	DCApriority	
int	GAPpriority	
int	SCPTpriority	
int	MaxTBMReq	
int	MaxDCAReq	
int	MaxGAReq	

Table A.53 EADSIM 'es_laymilid' data structure

Type	Variable Name	Remark
char	MilID	string size : 25
es_laymilid *	ES_NextLayMilIDAddr	structure pointer

Table A.54 EADSIM 'es_laytime' data structure

Type	Variable Name	Remark
int	OnTime	
int	OffTime	
es_laytime *	ES_NextlayTimeAddr	structure pointer

Table A.55 EADSIM 'es_laypointing' data structure

Type	Variable Name	Remark
char	ToPlatform	string size : 25
int	OnTime	
int	OffTime	
int	AntHeight	
float	VerAngle	
float	HorAngle	
unsigned char	PtMode	
es_laypointing *	ES_NextLayPointingAddr	structure pointer

Table A.56 EADSIM 'es_laynurdwaypoint' data structure

Type	Variable Name	Remark
float	Lat	
float	Lon	
float	Alt	
unsigned char	Type	
chat	MilID	string size : 25
unsigned char	TerrainFlag	boolean
unsigned char	Zmode	boolean
float	Speed	
int	OnTime	
int	OffTime	
es_laywaypoint *	ES_NextLayNURDWayPointAddr	structure pointer

Table A.57 EADSIM 'es_layaircraftdata' data structure

Type	Variable Name	Remark
char	Name	string size : 25
unsigned char	Domain	Enumeration Type
unsigned int	Country	Enumeration Type
unsigned char	Category	Enumeration Type
unsigned char	SubCategory	Enumeration Type
unsigned char	Specific	Enumeration Type

A.3 BATTLESIM

Table A.58 BATTLESIM 'bs_scenario' data structure

Type	Variable Name	Remark
BS_Header	Header	structure
bs_object *	NextObjAddr	structure pointer

Table A.59 BATTLESIM 'BS_TerrainMinCoord' data structure

Type	Variable Name	Remark
double	xmin	
double	ymin	
double	zmin	

Table A.60 BATTLESIM 'BS_TerrainMaxCoord' data structure

Type	Variable Name	Remark
double	xmax	
double	ymax	
double	zmax	

Table A.61 BATTLESIM 'bs_sector' data structure

Type	Variable Name	Remark
double	xmin	
double	ymin	
double	zmin	
double	xmax	
double	ymax	
double	zmax	
bs_sector *	NextSectorAddr	structure pointer

Table A.62 BATTLESIM 'bs_icon' data structure

Type	Variable Name	Remark
int	IconNumber	Enumeration Type
char	IconName	string size : 20
bs_icon *	NextIconAddr	structure pointer

Table A.63 BATTLESIM 'BS_Header' data structure

Type	Variable Name	Remark
char	VersionNumber	string size : 20
char	TerrainFileName	string size : 50
BS_TerrainMinCoord	TerrainMinCoord	structure
BS_TerrainMaxCoord	TerrainMaxCoord	structure
int	NumSectors	
bs_sector *	NextSectorAddr	structure pointer
int	NumIcons	
bs_icon *	NextIconAddr	structure pointer

Table A.64 BATTLESIM 'bs_object' data structure

Type	Variable Name	Remark
BS_DescObj	DescObj	structure
int	NumRoutePoints	
bs_routepoint *	NextRoutePoints	structure pointer
int	NumSensors	
bs_sensor *	NextSensorAddr	structure pointer
int	NumArmaments	
bs_armament *	NextArmamentAddr	structure pointer
int	NumTargets	
bs_target *	NextTargetAddr	structure pointer
int	NumDefensiveSys	
bs_defensive *	NextDefensiveSysAddr	structure pointer
bs_object *	NextObjAddr	structure pointer

Table A.65 BATTLESIM 'BS_ObjectLocation' data structure

Type	Variable Name	Remark
double	xcoord	
double	ycoord	
double	zcoord	

Table A.66 BATTLESIM 'BS_ObjectVelocity' data structure

Type	Variable Name	Remark
double	xvel	
double	yvel	
double	zvel	

Table A.67 BATTLESIM 'BS_ObjectOrientation' data structure

Type	Variable Name	Remark
double	yawrate	
double	pitchrate	
double	rollrate	

Table A.68 BATTLESIM 'BS_DescObj' data structure

Type	Variable Name	Remark
int	ObjectType	Enumeration Type
int	ObjectID	
double	CurrentTime	
BS_ObjectLocation	ObjectLoc	structure
BS_ObjectVelocity	ObjectVel	structure
BS_ObjectOrientation	ObjectOri	structure
double	PlayerSize	
double	mass	
int	ObjectLoyalty	
int	FuelStat	
int	Condition	
int	Vulnerability	
int	Experience	
int	ThreatKnow	
int	MinTurnRad	
int	MaxSpeed	
int	AvgFuelCons	
int	MaxClimb	

Table A.69 BATTLESIM 'bs_routepoint' data structure

Type	Variable Name	Remark
double	xcoord	
double	ycoord	
double	zcoord	
bs_routepoint *	NextRoutePointAddr	structure pointer

Table A.70 BATTLESIM 'bs_sensor' data structure

Type	Variable Name	Remark
int	SensorType	
int	SensorRange	
int	SensorResolution	
bs_sensor *	NextSensorAddr	structure pointer

Table A.71 BATTLESIM 'bs_armament' data structure

Type	Variable Name	Remark
int	ArmamentType	
int	ArmamentRange	
int	ArmamentYield	
int	ArmamentAccuracy	
int	ArmamentSpeed	
int	ArmamentCount	
bs_armament *	NextArmamentAddr	structure pointer

Table A.72 BATTLESIM 'bs_target' data structure

Type	Variable Name	Remark
int	TargetType	
double	TargetXCoord	
double	TargetYCoord	
double	TargetZCoord	
bs_target *	NextTargetAddr	structure pointer

Table A.73 BATTLESIM 'bs_defensive' data structure

Type	Variable Name	Remark
int	DefensiveSysType	
int	DefensiveSysRange	
int	DefensiveSysEffect	
bs_defensive *	NextDefensiveSysAddr	structure pointer

A.4 Transitional File

Table A.74 Transitional File 'tf_transfile' data structure

Type	Variable Name	Remark
char	Version	string size : 25
unsinged long	TF_NumEntities	
tf_database *	NextDatabaseAddr	structure pointer

Table A.75 Transitional File 'tf_database' data structure

Type	Variable Name	Remark
tf_entityheader *	TF_EntityHeaderAddr	structure pointer
tf_entitytype *	TF_EntitytTypeAddr	structure pointer
tf_entitylocation *	TF_EntityLocationAddr	structure pointer
tf_entityvelocity *	TF_EntityVelocityAddr	structure pointer
tf_entityorientation *	TF_EntityOrientationAddr	structure pointer
unsigned char	TF_NumKindMunitions	
tf_munitiontype *	NextMunitionAddr	structure pointer
unsigned char	TF_NumSensors	
tf_sensorid *	NextSensorAddr	structure pointer
unsigned char	TF_NumJammers	
tf_jammerid *	NextJammerAddr	structure pointer
unsigned char	TF_NumTasks	
tf_tasktype	NextTaskAddr	structure pointer
tf_database *	NextDatabaseAddr	structure pointer

Table A.76 Transitional File 'tf_point' data structure

Type	Variable Name	Remark
double	LocationX	
double	LocationY	
double	LocationZ	
tf_point *	NextPointAddr	structure pointer

Table A.77 Transitional File 'tf_location' data structure

Type	Variable Name	Remark
double	LocationX	
double	LocationY	
double	LocationZ	

Table A.78 Transitional File 'tf_entityheader' data structure

Type	Variable Name	Remark
unsigned int	EntityId	Enumeration Type
unsigned char	ForceId	Enumeration Type
char	EntityName	string size : 25

Table A.79 Transitional File 'tf_entitytype' data structure

Type	Variable Name	Remark
unsigned char	Domain	Enumeration Type
unsigned int	Country	Enumeration Type
unsigned char	Category	Enumeration Type
unsigned char	SubCategory	Enumeration Type
unsigned char	Specific	Enumeration Type

Table A.80 Transitional File 'tf_entityvelocity' data structure

Type	Variable Name	Remark
float	VelocityX	
float	VelocityY	
float	VelocityZ	

Table A.81 Transitional File 'tf_entityorientation' data structure

Type	Variable Name	Remark
float	Psi	
float	Theta	
float	Phi	

Table A.82 Transitional File 'tf_munitiontype' data structure

Type	Variable Name	Remark
unsigned char	Domain	Enumeration Type
unsigned int	Country	Enumeration Type
unsigned char	Category	Enumeration Type
unsigned char	SubCategory	Enumeration Type
unsigned char	Specific	Enumeration Type
unsigned int	Amount	
tf_munitiontype *	NextMunitionAddr	structure pointer

Table A.83 Transitional File 'tf_sensorid' data structure

Type	Variable Name	Remark
char	SensorID	string size : 25
tf_sensorid *	NextSensorAddr	structure pointer

Table A.84 Transitional File 'tf_jammerid' data structure

Type	Variable Name	Remark
char	JammerID	string size : 25
tf_jammerid *	NextJammerAddr	structure pointer

Table A.85 Transitional File 'tf_tasktype' data structure

Type	Variable Name	Remark
unsigned char	TaskType	Enumeration Type
void *	TaskDataAddr	address pointer
tf_tasktype *	NextTaskAddr	structure pointer

Appendix B. Enumeration Type for Task Type of Transitional Prototype

B.1 Task Name

Field Value 1-99 : Ground Mission

Field Value 100-149 : Air Mission

100-124 : FWA Mission

125 - 149 : RWA Mission

Field Value 150-199 : Surface Mission

Field Value 200-249 : Space Mission

Field Value 250-255 : Extra

Table B.1 Enumeration Type of Task Name

Field Value	Task Kind
0	Others
1	Move
2	Road March
3	Follow a Vehicle
4	Follow Simulator
5	Pursue
6	Hasty Occupy Position
7	Assault
8	Traveling Overwatch
9	Overwatch Movement
10	Withdraw
11	Breach
12	Attack By Fire
13	Concealment
14	Delay
15	Repair
16	Sevice Station
17	Cross-leveling
18	Change Formation
19	Rendezvous
100	FWA Sweep
101	FWA CAP
102	FWA CAS Mission
103	FWA Ingress
104	FWA Attack Ground Target

105	FWA Egress
106	FWA Return to Base
107	FWA Interdiction Mission
125	RWA Fly Route
126	RWA Hover
127	RWA Orbit
128	RWA Assemble
129	RWA Attack
130	RWA Hasty Occupy Position

B.2 Data Structures for Each Task Type

B.2.1 Move

Table B.2 Task 'Move' data structure

Type	Variable Name	Remark
unsigned char	NumPoints	Route
tf_point *	PointAddr	
unsigned char	TravelType	Enumeration Type
unsigned char	Formation	Enumeration Type
float	RateOfMarch	
float	CatchUpSpeed	
float	DismountedSpeed	
unsigned int	FollowingLeader	
unsigned int	FollowingLeaderDegree	
float	FollowingLeaderDistance	
unsigned char	NumPoints	LeftBoundary
tf_point *	PointAddr	
unsigned char	NumPoints	RightBoundary
tf_point *	PointAddr	
unsigned char	Spacing	Enumeration Type
float	UserSpecifiedSpacing	
float	IFVOffsetX	
float	IFVOffsetY	

B.2.2 Road March

same as Task "Move".

B.2.3 Follow a Vehicle

same as Task "Move".

B.2.4 Follow Simulator

Table B.3 Task 'Follow Simulator' data structure

Type	Variable Name	Remark
unsigned char	SimulatorToFollow	
unsigned char	TravelType	Enumeration Type
unsigned char	Formation	Enumeration Type
unsigned char	Spacing	Enumeration Type
float	UserSpecifiedSpacing	
float	RejoinSimDistance	

B.2.5 Pursue

same as Task "Move".

B.2.6 Hasty Occupy Position

Table B.4 Task 'Hasty Occupy Position' data structure

Type	Variable Name	Remark
unsigned char	NumPoints	BattlePosition
tf_point *	PointAddr	
unsigned char	NumPoints	EngagementArea
tf_point *	PointAddr	
tf_location *	LeftTRP	
tf_location *	RightTRP	
float	Speed	
tf_location *	TriggerLine	
unsigned char	TriggerCriteria	
unsigned char	TriggerUnitSize	
unsigned char	NumPoints	SecondaryBattlePosition
tf_point *	PointAddr	
unsigned char	NumPoints	SecondaryEnggArea
tf_point *	PointAddr	
tf_location *	SecondaryLeftTRP	
tf_location *	SecondaryRightTRP	

B.2.7 Assault

Table B.5 Task 'Assault' data structure

Type	Variable Name	Remark
unsigned char	NumObjectives	Objectives
tf_point *	ObjectivesAddr	
unsigned char	NumPoints	Route
tf_point *	PointAddr	
float	Speed	
float	DismountedSpeed	
float	StoppingAssaultCriteria	
unsigned char	SecureObjective	Enumeration Type
unsigned char	Formation	Enumeration Type
unsigned char	Spacing	Enumeration Type
float	UserSpecifiedSpacing	
float	IFVOffsetX	
float	IFVOffsetY	

B.2.8 Traveling Overwatch

Table B.6 Task 'Traveling Overwatch' data structure

Type	Variable Name	Remark
unsigned char	NumPoints	Route
tf_point *	PointAddr	
float	RateOfMarch	
float	CatchUpSpeed	
float	SupportGroupFollowingDistance	
unsigned char	Formation	Enumeration Type
unsigned char	Spacing	Enumeration Type
float	UserSpecifiedSpacing	
unsigned char	ConformToTerrain	boolean

B.2.9 Overwatch Movement

Table B.7 Task 'Overwatch Movement' data structure

Type	Variable Name	Remark
unsigned char	NumPoints	Route
tf_point *	PointAddr	
unsigned char	UseConcealedRoute	boolean
unsigned char	NumPoints	LeftBoundary
tf_point *	PointAddr	
unsigned char	NumPoints	RightBoundary
tf_point *	PointAddr	
unsigned char	Movement	Enumeration Type
float	RateOfMarch	

float	CatchUpSpeed	
float	DismountedSpeed	
unsigned int	FollowingLeader	
unsigned int	FollowingLeaderDegree	
float	FollowingLeaderDistance	
unsigned char	Spacing	Enumeration Type
float	UserSpecifiedSpacing	
unsigned char	Formation	Enumeration Type
unsigned char	DIFormation	
float	IFVOffsetX	
float	IFVOffsetY	

B.2.10 Withdraw

Table B.8 Task 'Withdraw' data structure

Type	Variable Name	Remark
unsigned char	NumPoints	Route
tf_point *	PointAddr	
unsigned char	Smoke	Enumeration Type
float	Speed	
float	SpeedLimit	

B.2.11 Breach

Table B.9 Task 'Breach' data structure

Type	Variable Name	Remark
unsigned char	NumPoints	Route
tf_point *	PointAddr	
unsigned char	CreateMarkers	boolean
float	MaxDistBtwMarks	
float	BreachLaneWidth	

B.2.12 Attack By Fire

same as Task "Hasty Occupy Position".

B.2.13 Concealment

Table B.10 Task 'Concealment' data structure

Type	Variable Name	Remark
unsigned char	NumPoints	Route
tf_point *	PointAddr	
float	Distance	
float	RateOfMarch	
float	FrontWidth	
unsigned int	ForwardSpread	
unsigned int	BackwardSpread	

B.2.14 Delay

Table B.11 Task 'Delay' data structure

Type	Variable Name	Remark
unsigned char	NumPoints	BattlePosition1
tf_point *	PointAddr	
unsigned char	NumPoints	BattlePosition2
tf_point *	PointAddr	
unsigned char	NumPoints	BattlePosition3
tf_point *	PointAddr	
unsigned char	NumPoints	BattlePosition4
tf_point *	PointAddr	
float	Speed	

B.2.15 Repair

Table B.12 Task 'Repair' data structure

Type	Variable Name	Remark
tf_location *	UMCPSite	
unsigned char	TowVehicle	
unsigned char	VehicleToRepair	

B.2.16 Sevice Station

Table B.13 Task 'Service Station' data structure

Type	Variable Name	Remark
unsigned char	SupplyUnit	

B.2.17 Cross-leveling

Table B.14 Task 'Cross Leveling' data structure

Type	Variable Name	Remark
tf_location *	CrossLevelingAreaLocation	
float	CrossLevelingAreaRadius	
unsigned char	SupplyToCrossLevel	Enumeration Type

B.2.18 Change Formation

Table B.15 Task 'Change Formation' data structure

Type	Variable Name	Remark
tf_location *	Destination	
unsigned char	Formation	Enumeration Type
unsigned int	FormationAngle	

B.2.19 Rendezvous

Table B.16 Task 'Rendezvous' data structure

Type	Variable Name	Remark
unsigned char	Partner	
tf_location *	Objective	
unsigned char	PUnitFormation	Enumeration Type
unsigned char	SUnitFormation	Enumeration Type
unsigned int	RelativePosition	
float	ParkingOffset	
float	MaxSpeed	

B.2.20 FWA Sweep

Table B.17 Task 'FWA Sweep' data structure

Type	Variable Name	Remark
unsigned char	NumPoints	Route
tf_point *	PointAddr	
float	Speed	
float	Altitude	
unsigned char	RadarSearchMode	Enumeration Type
unsigned char	RadarVolumeAzimuth	Enumeration Type
unsigned char	RadarVolumeElevation	Enumeration Type
unsigned char	RadarVolumeRange	Enumeration Type

unsigned char	RadarVcSetting	Enumeration Type
unsigned char	RadarOrientationType	Enumeration Type
tf_location *	RadarOrientationLocation	
unsigned int	RadarOrientationAzimuth	
unsigned int	RadarOrientationElevation	

B.2.21 FWA CAP

Table B.18 Task 'FWA CAP' data structure

Type	Variable Name	Remark
tf_location *	CAPPosition	
unsigned int	CAPOrientation	
float	LenghOfLegs	
float	InboundLegSpeed	
float	OutboundLegSpeed	
float	CAPAltitude	
unsigned char	RadarSearchMode	Enumeration Type
unsigned char	RadarVolumeAzimuth	Enumeration Type
unsigned char	RadarVolumeElevation	Enumeration Type
unsigned char	RadarVolumeRange	Enumeration Type
unsigned char	RadarVcSetting	Enumeration Type
unsigned char	RadarOrientationType	Enumeration Type
tf_location *	RadarOrientationLocation	
unsigned int	RadarOrientationAzimuth	

B.2.22 FWA CAS Mission

Table B.19 Task 'FWA CAS Mission' data structure

Type	Variable Name	Remark
tf_location *	ForwardAirController	
unsigned char	NumPoints	RouteToForwardAirController
tf_point *	PointAddr	
unsigned char	NumPoints	OptionalReturnRoute
tf_point *	PointAddr	
float	FlightSpeed	
float	Altitude	
unsigned char	FlightMethod	Enumeration Type
unsigned char	Formation	Enumeration Type
unsigned int	TimeOnStation	
unsigned char	ActionsAfterMission	Enumeration Type

B.2.23 FWA Ingress

Table B.20 Task 'FWA Ingress' data structure

Type	Variable Name	Remark
unsigned char	NumPoints	Route
tf_point *	PointAddr	
float	Speed	
float	Altitude	
unsigned char	Formation	Enumeration Type
unsigned char	MovementType	Enumeration Type
unsigned char	AtEndOfRoute	Enumeration Type

B.2.24 FWA Attack Ground Target

Table B.21 Task 'FWA Attack Ground Target' data structure

Type	Variable Name	Remark
unsigned char	NumTargets	
tf_point *	TargetAddr	
unsigned char	NumPoints	Route
tf_point *	PointAddr	
float	Speed	
float	Altitude	
unsigned char	MovementType	Enumeration Type
unsigned char	Formation	Enumeration Type
unsigned char	AttackGeometry	Enumeration Type
unsigned char	AttackEntry	Enumeration Type
unsigned char	AttackDelivery	Enumeration Type
weaponsenabled *	WeaponEnabledAddr	structure pointer
unsigned char	MissionType	Enumeration Type

Table B.22 Task 'weaponsenabled' data structure

Type	Variable Name	Remark
unsigned char	Bombs	boolean
unsigned char	Guns	boolean
unsigned char	Missiles	boolean

B.2.25 FWA Egress

same as Task "FWA Ingress".

B.2.26 FWA Return to Base

Table B.23 Task 'FWA Return to Base' data structure

Type	Variable Name	Remark
unsigned char	NumPoints	Route
tf_point *	PointAddr	
unsigned char	NumPoints	Optional Return Route
tf_point *	PointAddr	
tf_location *	TargetLocation	
unsigned char	FlightMethod	Enumeration Type
float	FlightSpeed	
float	Altitude	
unsigned char	Formation	Enumeration Type
unsigned char	MissionType	Enumeration Type
unsigned char	ActionAfterMission	Enumeration Type

B.2.27 FWA Interdiction

Table B.24 Task 'FWA Interdiction' data structure

Type	Variable Name	Remark
unsigned char	NumPoints	RouteToTarget
tf_point *	PointAddr	
unsigned char	NumPoints	OptionalReturnRoute
tf_point *	PointAddr	
tf_location *	TargetLocation	
unsigned char	FlightMethod	Enumeration Type
float	FlightSpeed	
float	Altitude	
unsigned char	Formation	Enumeration Type
unsigned char	MissionType	Enumeration Type
unsigned char	ActionAfterMission	Enumeration Type

B.2.28 RWA Fly Route

Table B.25 Task 'RWA Fly Route' data structure

Type	Variable Name	Remark
unsigned char	NumPoints	Route
tf_point *	PointAddr	
float	Speed	
float	CatchUpSpeed	
float	Altitude	
unsigned char	Formation	Enumeration Type
unsigned char	Spacing	Enumeration Type
float	UserDefinedSpacing	
unsigned char	MovementType	Enumeration Type

unsigned char	TargetFacingMode	boolean
tf_location *	NapOfEarthTRP	

B.2.29 RWA Hover

Table B.26 Task 'RWA Hover' data structure

Type	Variable Name	Remark
tf_location *	FARPPoint	
float	Speed	
float	Altitude	
unsigned char	ActiveFuelFARP	boolean
unsigned char	ActiveAmmoFARP	boolean

B.2.30 RWA Orbit

Table B.27 Task 'RWA Orbit' data structure

Type	Variable Name	Remark
tf_location *	CenterPoint	
float	Radius	
float	Speed	
float	Altitude	

B.2.31 RWA Assemble

same as Task "RWA Orbit".

B.2.32 RWA Attack

Table B.28 Task 'RWA Attack' data structure

Type	Variable Name	Remark
tf_location *	Objective	
float	Speed	
unsigned char	AttackType	Enumeration Type

B.2.33 RWA Hasty Occupy Position

Table B.29 Task 'RWA Hasty Occupy Position' data structure

Type	Variable Name	Remark
unsigned char	NumPoints	BattlePosition
tf_point *	PointAddr	
tf_location *	LeftTRP	

tf_location *	RightTRP	
tf_location *	EnggAreaTRP	
unsigned char	TargetFacingMode	boolean
float	Speed	
float	Altitude	
float	AltitudeAtEnt	

B.3 Enumeration Type for Variables in Task

Task Kind	Variable Names	Field Value	
Assault	Secure Objective	0: No 1: Yes*	*: default value
	Formation	0 : Other	
		1 : Column	
		2 : Staggered-Column	
		3 : Echelon-Right	
		4 : Echelon-Left	
		5 : Line*	
		6 : Wedge	
		7 : Vee	
	Spacing	0 : Other	
		1 : Closed*	
		2 : Open	
		3 : User Specifed	
Breach	Create Markers	0 : Other	
		1 : Breach Lanes*	
		2 : No Markers	
Change Formation	Formation	*Same as 'Assault'-'Formation'*	*Wedge
Cross-leveling	Supply to Cross-level		
		0 : Other	
		1 : Most unbalanced non-fuel munition*	
		2 : Fuel	
Follow Simulator	Travel Type	0 : Other	
		1 : Road March	
		2 : Cross Country*	
	Formation	*Same as 'Assault'-'Formation'*	*Wedge
	Spacing	*Same as 'Assault'-'Spacing'*	*Closed
FWA Attack Ground Tgt	Movement Type	0 : Other	
		1 : Low Level	
		2 : Contour*	
	Formation	0 : Other	
		1 : Fighting Wing*	
		2 : Line Abreast	
		3 : Bearing	

		4 : Trail
		5 : VIC
		6 : Box
		7 : Offset Box
	Attack-Geometry	0 : Other
		1 : Direct*
		2 : Split
		3 : 90-10
		4 : Trail
	Attack-Entry	0 : Other
		1 : Pop-Up
		2 : Level*
		3 : Standoff AGM Pop-Up
		4 : Standoff AGM Med Alt Dive
	Attack-Delivery	0 : Other
		1 : Laydown
		2 : Low Altitude Dive*
		3 : Medium Altitude Dive
		4 : Strafe
	Weapons Enable	
	Bombs:	0:No 1:Yes*
	Guns :	0:No 1:Yes*
	Missiles :	0:No 1:Yes*
	Mission Type	0 : Other
		1 : Targets are vehicles*
		2 : Target is a location
FWA CAP	Radar Search Mode	0 : Other
		1 : Track While Scan(Manual)*
		2 : Track While Scan(Auto)
	Radar Volumn Azimuth	0 : Other
		1 : +/- 10 Degrees
		2 : +/- 20 Degrees
		3 : +/- 40 Degrees*
		4 : +/- 60 Degrees
	Radar Volumn Elevation	0 : Other
		1 : 8 Bar
		2 : 4 Bar
		3 : 2 Bar*
		4 : 1 Bar
	Radar Volumn Range	0 : Other
		1 : 20 NM
		2 : 50 NM
		3 : 100 NM*
		4 : 200 NM
	Radar Vc Setting	0 : Other
		1 : +/- 1600 knots*
		2 : 0-1600 knots
		3 : 0-(-1600) knots
	Radar Orientation Type	0 : Other
		1 : Use Orientation Az/El *
		2 : Use Orientation Location

FWA CAS	FlightMethod	*Same as 'FWA AGT'-'Movement Type'* *Contour
	Formation	*Same as 'FWA AGT'-'Formation'* *Fighting Wing
	At End of Route	0 : Other 1 : Orbit* 2 : Land
FWA Sweep	Radar Search Mode	*Same as 'FWA CAP'*
	Radar Volumn Azimuth	*Same as 'FWA CAP'*
	Radar Volumn Elevation	*Same as 'FWA CAP'*
	Radar Volumn Range	*Same as 'FWA CAP'*
	Radar Vc Setting	*Same as 'FWA CAP'*
	Radar Orientation Type	*Same as 'FWA CAP'*
FWA Ingress	Formation	*Same as 'FWA AGT'-'Formation'* *Fighting Wing
	Movement Type	*Same as 'FWA AGT'-'Movement Type'* *Contour
	At End of Route	0 : Other 1 : Orbit* 2 : Land
FWA Interdiction *Contour	FlightMethod	*Same as 'FWA AGT'-'Movement Type'*
	Formation	*Same as 'FWA AGT'-'Formation'* *Fighting Wing
	Mission Type	*Same as 'FWA AGT'-'Mission Type'* *Attack fixed emplacement
	ActionsAfterMission	0 : Other 1 : Orbit* 2 : Land
Occupy Position	Use Alternative Position	0:No 1:Yes*
	Trigger Criterion	0 : Other 1 : First Vehicle* 2 : Last Vehicle 3 : Unit Center of Mass
	Trigger Unit Size	0 : Other 1 : Vehicle 2 : Squad 3 : Section 4 : Platoon * 5 : Company 6 : Battallion 7 : Regiment 8 : Brigade 9 : Division 10 : Corps 11 : Army 12 : Army Group
Travel	Travel Type	*Same as 'Follow Simulator'*
	Formation	*Same as 'Assault'*
	Spacing	*Same as 'Assault'*

Overwatch Move	Use Concealed Routes	0:No 1:Yes*
	Movement	0 : Other
		1 : Boundary Overwatch*
		2 : Non-Boundary Overwatch
		3 : Boundary(No Overwatch)
	Spacing	*Same as 'Assault'*
	Formation	0 : Other
		1 : Line*
		2 : Column
	DI Formation	0 : Other
		1 : Rudel*
		2 : Reihe
Randezvous	PUnitFormation	0 : Other
		1 : Wedge
		2 : Line
		3 : Vee
		4 : Column
		5 : Staggered-Column*
		6 : Echelon-Right
		7 : Echelon-Left
	SUnitFormation	0 : Other
		1 : Wedge
		2 : Line
		3 : Vee
		4 : Column
		5 : Staggered-Column*
		6 : Echelon-Right
		7 : Echelon-Left
RWA Attack	Attack Type	0 : Other
		1 : Hover Attack*
		2 : Running Attack
RWA Fly Route	Fomation	0 : Other
		1 : Wedge*
		2 : Line
		3 : Echelon Right
		4 : Echelon Left
		5 : Trail
		6 : Straggered Right
		7 : Straggered Left
		8 : Pair in trail
	Spacing	*Same as 'Assault'*
	Movement Type	0 : Other
		1 : Nap of Earth
		2 : Contour*
		3 : Low Level
	Target Facing Mode	0:No* 1:Yes
RWA Occupy Position	Target Facing Mode	0:No* 1:Yes
RWA Unit FARP	Active Fuel FARP	0 : Other

		1 : Disable Reaction
		2 : Enable Reaction*
	Active Ammo FARP	0 : Other
		1 : Disable Reaction
		2 : Enable Reaction*
Traveling Overwatch	Formation	*Same as 'Assault'*
	Spacing	*Same as 'Assault'*
	Conform to Terrain	0:No* 1:Yes
Withdraw	Smoke	0 : Other
		1 : Enabled
		2 : Disabled*

Appendix C. Command Line Interface for SFT

NAME

SFT - Scenario File Translator

SYNOPSIS

SFT [-help] [-N|n] [-C|c] [options]

DESCRIPTION

SFT converts a scenario to another type of scenario. A scenario file that will be translated to another format is called 'source scenario', and a scenario file which is produced by SFT is 'target scenario'.

A 'source simulation' is a computer simulation name which operates a source scenario, a 'target simulation' is a name of computer simulation which the target scenario will be run.

OPTIONS

The following options are supported:

-help
: Show the usage message.

-N|n
: Show the corresponding numbers of each simulation

-C|c
: SFT interactive with the user. SFT asks a source simulation type, a source scenario file name, a target simulation type and a target scenario file name.

-s [number] [input]
: Name the source simulation name number and source scenario file. 'input' should have a correct directory path and file name.

-t [number] [output]
: Name the target simulation name number and target scenario file. 'output' should have a correct directory path and file name.

OTHER NECESSARY INPUT

When converting a source scenario to ModSAF scenario, Exercise ID and terrain file path/name are needed.

Converting to EADSIM scenario file, laydown file name without suffix (.lay) is necessary.

If a source simulation is BATTLESIM, there can be several input files to build a scenario. Be sure all input files are ended with series of number from 0. In that case, input file to SFT is the first file name and the number of files is needed to convert to a target scenario.

EXAMPLES

```
SFT -s 1 ./data/scn.1 -t 2 ./data/scn.2
```

Convert a source scenario scn.1 of corresponding source simulation, which is located in ./data to a target scenario scn.2 of simulation number 2 into ./data directory.

AUTHOR

Captain Heon-Gyu Park, Republic of Korea, Army.

Bibliography

[Adam93] Adams, C. "DOD Embraces Warfighting Simulation," Federal Computer Week, 7:8-10 (November 1993).

[ADST95] Advanced Distributed Simulation Technology, USER'S MANUAL for ModSAF, Contract N61339-91-D-0001. 28 April 1995.

[Army94] Secretary of Army, Army Model and Simulation Master Plan, 3 May 1994.

[Bone90] Boner, K.E., D.R. Hardy and T.R. Tiernan. "Battle Force Inport/Simulation Networking: SIMNET Protocol Suitability Consideration." Technical Note 1614, Naval Ocean Systems Center, San Diego CA. June 1990.

[Cour95] Courtemanche, Anthony J., Andy Ceranowicz. "ModSAF Development Status." Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation. Institute for Simulation & Training, May 1995.

[Cera95] Ceranowicz, Andy. Modular Semi-Automated Forces, Loral Advanced Distributed Simulation, Cambridge, MA, 1995.

[DIS94] DIS Steering Committee. The DIS Vision: A Map to the Future of Distributed Simulation, Version 1, May 1994.

[Dod95] Secretary of Defense for Acquisition and Technology, Modeling and Simulation (M&S) Master Plan. Department of Defense, October 1995.

[Fuji93] Fujimoto, Richard M. "Parallel Discrete Event Simulation: Will the Field Survive," ORSA Journal on Computing. Summer 1993.

[Gard93] Gardner, Michael T. A Distributed Interactive Simulation Based Remote Debriefing Tool for Red Flag Missions. MS thesis, AFIT/GCS/ENG/93-09, School of Engineering, Air Force Institute of Technology(AU), Wright-Patterson Air Force Base OH, December 1993.

[Garr95] Garrett, Randy. "A New Simulation Paradigm: Advanced Distributed Simulation," PHALANX, 25-27 September 1995

[Harv91] Harvey, Edward P. and Richard L. Schaffer. "The Capability of the Distributed Interactive Simulation Networking Standard to Support High Fidelity Aircraft Simulation." Proceedings of the 13th Interservice/Industry Training Systems Conference. 263-271, 1991.

[Hill94] Hiller, James B. Analytic Performance Models Of Parallel Battlefield Simulation Using Conservative Processor Synchronization. MS thesis, AFIT/GCS/ENG/94D-08, School of Engineering, Air Force Institute of Technology(AU), Wright-Patterson Air Force Base OH, December 1994.

[IST93] Institute for Simulation and Training, Standard for Information Technology -- Protocols for Distributed Interactive Simulation Application. Version 2.0 Third draft, Technical Report IST-CR-93-15, University of Florida, May 1993.

[IST94] Institute for Simulation and Training, Standard for Distributed Interactive Simulation -- Application Protocols. Version 2.0 Fourth draft, Technical Report IST-CR-93-40, University of Florida, March 1994.

[Jone96] Jones, Anita K. "ADS for Analysis – Challenges for the Future," PHALANX, June 1996.

[McDo91] McDonald, L. Bruce, et al. "Standard Protocol Data Units for Entity Information and Interaction in a Distributed Interactive Simulation." Proceedings of the 13th Interservice / Industry Training Systems Conference. 119-126, 1991.

[Mill89] Miller, Harold G. "Wargaming Networks for Training." Proceedings Interactive Networked Simulation for Taining. 44, 1989.

[Neel87] Neelamkavil, Francis. "Computer Simulation and Modelling." John wiley & sons. 1987.

[Pick95] Pickett, H. Kent, Mikel D. Petty. "Report on The Stat of Computer Generated Forces 1994." Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation. Institute for Simulation & Training, May 1995.

[Pope91] Pope, Arthur R. "The SIMNET Network and Protocols." Bolt Beranek and Newman Inc. Report Number 7627 prepared for Defense Advanced Research Projects Agency, June 1991.

[Shea92] Sheasby, Steven M. Management of SIMNET and DIS Entities in Synthetic Environments. MS thesis, AFIT/GCS/ENG/92D-16, School of Engineering, Air Force Institute of Technology(AU), Wright-Patterson Air Force Base OH, December 1992.

[Siko95] Sikora, Jim and Phil Coose. "What in the World is ADS?" PHALANX, 1, 6-8 June 1995

[Stev90] Stevens, W. Richard. Unix Network Programming. Englewood Cliffs, New Jersey: Prentice Hall, 1990.

[TBE95] Teledyne Brown Engineering, USER'S MANUAL for EADSIM, 26 November 1995.

[Thor88] Thorpe, Jack A. "Warfighting with SIMNET-A Report from the Front." Proceedings of the 10th Interservice/Industry Training Systems Conference. 127-135, 1988.

[Webs94] Merriam Webster's Collegiate Dictionary - 10th Edition. Merriam-Webster, Incorporated, Messachusetts, 1994

Vita

Captain Heon-Gyu Park was born on April 7, 1966, in YaeGwan, GyungSangBukDo, Republic of Korea. He graduated from the Korea Military Academy with a Bachelor of Science degree in Computer Engineering in March 1988. After receiving his commission as an Army Infantry officer, he was a platoon leader at GOP (Guard of Post) near PanMunJom from 1988 through 1989. His next assignment was a leader officer of a guard of honor in the Third Army Headquarters. In 1991, after completing the Officer Basic Course (OBC) of Computer Engineering, he was assigned to the 1st Anti-Air Brigade as a Computer Engineer. There he led a team of computer software engineers and was responsible for maintaining all systems and the network. He remained in that position until May of 1994 when he was selected to attend the Air Force Institute of Technology at Wright-Patterson Air Force Base, Dayton Ohio to study for his Master of Science degree in Electrical and Computer Engineering. After completing his Masters degree, he will be assigned to the Headquarters of Army, Republic of Korea.

Permanent Address:
33/4 316-115 JeonPo 1 Dong
Pusan City, Republic of Korea
614-041

Tel:
Int'l: 82-51-809-9772
Korea: 051-809-9772

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1996	Master's Thesis

4. TITLE AND SUBTITLE

THE DEVELOPMENT OF A SCENARIO TRANSLATOR FOR DISTRIBUTED SIMULATIONS

5. FUNDING NUMBERS

6. AUTHOR(S)

Heon-Gyu Park

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology, WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/96D-22

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

There exists a variety of simulation generation and analysis products which have differing purposes and functions designed to simulate a real military battlefield. Due to the particular purpose of each simulation, it is impractical to use one scenario of a simulation directly with another simulation without a translator since there is no standard scenario format. In the current environment, interoperability between simulations is becoming more important in large scale simulations and distributed exercises. The Scenario File Translator (SFT) provides an easy and accurate way to create a scenario from a heterogeneous simulation. The SFT can load and save the three research simulations: ModSAF, EADSIM, and BATTLESIM. It also defines the general transitional prototype (TP) which is the information most commonly used to create a mock battlefield computer simulation. Every source scenario is converted through TP for program extensibility and reusability. System functionality is accessible through a graphical user interface (GUI). Although this system was designed for three simulations, it can be applied to any other simulations by creating only two additional functions: one which maps the new scenario to the TP and another which remaps the TP into the transitional scenario protocol.

14. SUBJECT TERMS

Scenario, Simulation, Translation, Conversion, DIS, Distributed Interactive Simulation, CGF, Computer Generated Forces, Mission Analysis

15. NUMBER OF PAGES

140

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL