1-1-1993

Integrated Eagle/BDS-D: Interim Report 2

Tracy R. Tolley

INSTITUTE FOR SIMULATION AND TRAINING

Institute for Simulation and Training
Computer Generated Forces Laboratory

November 30, 1993

Integrated Eagle/BDS-D Interim Report 2

Institute for Simulation and Training
3280 Progress Drive
Orlando FL 32826

University of Central Florida
Division of Sponsored Research

IST-CR-93-44

INSTITUTE FOR SIMULATION AND TRAINING

Integrated Eagle/BDS-D

Interim Report 2

IST-CR-93-44
November 30, 1993

Prepared by

Clark Karr

Reviewed by

Mikel Petty

Mikel D Petty

Institute for Simulation and Training
Computer Generated Forces Laboratory
3280 Progress Drive • Orlando, Florida 32826

University of Central Florida • Division of Sponsored Research

Table of Contents

1. Introduction

1.1 Purpose

The first Engineering Change Order (ECP1) to STRICOM contract N61339-92-K-0002, "Integrated Eagle/BDS-D" requires quarterly Interim Progress Reports. This Interim Progress Report covers the period 93/07/15 through 93/11/15 and is deliverable A007.

This document contains some text from earlier Technical Reports:

IST-TR-92-12 - "Integrated Eagle BDS-D Demonstrations: Aggregation, Operation Orders/ Operator Intent, and Indirect Fire"

IST-CR-93-24 - "Experiments in Incorporating Constructive/Virtual Simulation in the Design and Development of Weapon Systems"

IST-CR-93-26 - "Integrated Eagle/BDS-D Work Plan"

IST-CR-93-31 - "Integrated Eagle/BDS-D Interim Report 1"

This text is repeated here for clarity and completeness.

1.2 Background - The Integrated Eagle/BDS-D Project

1.2.1 Project Overview

The Integrated Eagle/BDS-D project is being performed by the Institute for Simulation and Training (IST) under contract to the US Army Simulation, Training, and Instrumentation Command (STRICOM). The project's contract number is N61339-92-K-0002. The STRICOM Contracting Office Technical Representative (COTR) for the project is Robert L. Paulson and the IST Principal Investigator (PI) is Clark R. Karr. The project began at IST on 19 June 1992.

The goal of the Integrated Eagle/BDS-D project is to demonstrate the feasibility of integrating constructive and virtual combat simulations of different granularities or representational scales. Constructive simulations are typically "aggregate" level, man-out- of-the-loop simulations while virtual simulations are typically "entity-level", man-in-the-loop simulations.

"Aggregate" battlefield simulations control groups of entities (e.g. the tanks in a tank company) as an aggregate rather than as a set of individual simulated entities. The position, movement, status, and composition of aggregate units are maintained for the unit as a whole and are the result of statistical analysis of the units' actions rather than the result of the actions of the individual entities. The Eagle system is one such aggregate simulation and was developed by TRAC. Eagle simulates ground combat at the company and battalion level and is used for combat development studies; it runs faster than real-time.

In contrast, "entity level" simulations represent each vehicle as a distinct simulation entity. The SIMNET networked training system and its DIS successors are examples of entity level simulations operating in real-time. Usually each "entity level" simulator represents a single tank or vehicle. They interact in a common simulated battlefield by exchanging information packets on the network that connects them. Under contract to STRICOM, IST has developed a Computer Generated Forces (CGF) Testbed which generates and controls multiple individual entities in a simulated battlefield on a single computer system. The CGF Testbed is briefly described in the next section. Battlefield Distributed Simulation-Developmental (BDS-D) is the class of SIMNET systems involved in experimental and developmental work. SIMNET and BDS-D will be used interchangeably throughout this document.

The Integrated Eagle/BDS-D project's goal is the integration of the Eagle and SIMNET/BDS-D

simulations. The lessons learned in connecting Eagle to BDS-D will be applicable in the DIS environment and applicable to other aggregate and entity level simulations. The results of this project will benefit both the training and analysis communities. The training community will benefit from the ability to conduct small unit (company and platoon) training exercises in the context of a larger Corps battle. The actions of the trainees can occur in the virtual training environment but affect and be affected by the course of the larger battle. The analytic community will benefit from having a mechanism for increasing the resolution of the aggregate level simulations. The increased resolution can provide information and detail which can then be abstracted into the aggregate level simulations.

As part of this project, mechanisms have been developed at IST whereby Eagle aggregate units (companies) are "disaggregated" into their component vehicles in the entity level environment. Disaggregation occurs when Eagle units enter a preregistered portion of the battlefield that is also represented as a high-resolution SIMNET/BDS-D polygonal terrain database. Disaggregation involves entity instantiation based on the current components of the unit, placement on the high-resolution terrain, and activation in the entity level battlefield.

Within the entity level battlefield, the individual disaggregated vehicles are controlled by the IST CGF Testbed. The position, movement, status, and composition of the aggregate unit in the Eagle simulation becomes a function of the position, movement, and survival of its component CGF entities.

While Eagle units are disaggregated, the Eagle simulation shifts to real-time execution to receive and incorporate the CGF entity information being generated in real-time. Direct fire combat occurs in the BDS-D environment between individual CGF entities.

Additionally, the human CGF controller can request indirect fire from aggregate Eagle artillery units. The Eagle artillery units produce indirect fire volleys that appear in BDS-D as individual detonations which damage/kill nearby CGF vehicles and fireteams.

Re-aggregation of CGF entities into their Eagle aggregate units occurs when the center of mass of each disaggregated unit moves outside the preregistered high-resolution terrain area.

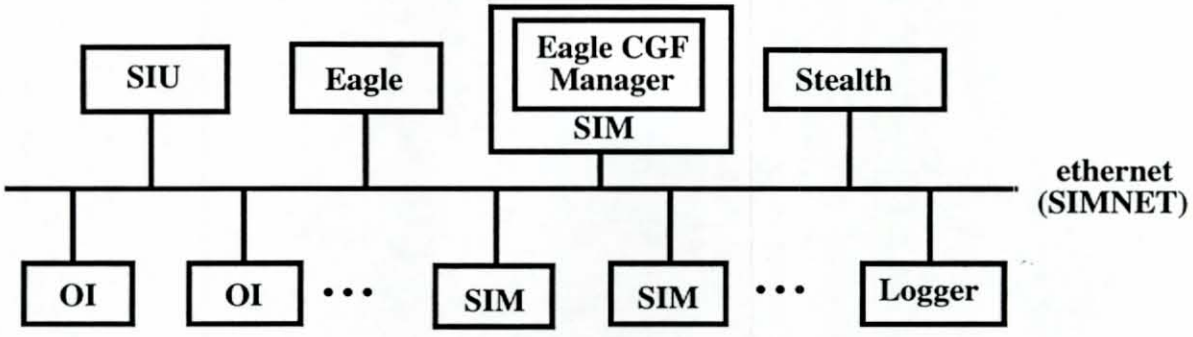
1.2.2 Project Components

Working on the overall Integrated Eagle/BDS-D project with IST are research groups at the US Army TRADOC Analysis Command (TRAC), Ft. Leavenworth KS, led by Kent Pickett, and at the Los Alamos National Laboratory (LANL), Los Alamos NM, led by Randy Michelsen.

TRAC is responsible for modifying the Eagle system as needed for this project. The TRAC modifications to Eagle include initiating disaggregation and re-aggregation events, responding to indirect fire request, and real-time execution of the corps/division/brigade/battalion command posts and maneuver units.

LANL is developing the Simulation Interface Unit (SIU), which is a software module that serves as the interface between the aggregate Eagle battlefield and the entity level SIMNET battlefield. Based on events in one environment, the SIU generates network packets that represent those events or their consequences in the other environment. For example, the SIU mediates the disaggregation of an Eagle company by sending a *disaggregation request* packet to the CGF Testbed. In the other direction, the SIU as a SIMNET node receives all SIMNET PDUs and abstracts and consolidates their information into the form needed by Eagle. The SIU processes standard SIMNET version 6.6.1 protocol data units (PDUs). In addition, the SIU and CGF Testbed exchange Interoperability Protocol (IOP) format packets, where IOP is a network protocol developed by IST and LANL specifically for this project.

As part of the Integrated Eagle/BDS-D project, a new software component, the Eagle CGF Manager, has been added to the IST CGF Testbed to coordinate all communications and activities between the IST CGF Testbed and the SIU. For example, disaggregation requests are sent from the SIU to the Eagle CGF Manager which interprets the requests and causes the disaggregation to take place in the virtual environment. When the disaggregation is complete, the Eagle CGF Manager communicates the results to the SIU.



Eagle	- Eagle Model
SIU	- Simulation Integration Unit
Eagle CGF Manager	- IST CGF Testbed Manager
OI	- IST CGF Testbed Operator Interface
SIM	- IST CGF Testbed Simulator
Logger	- IST CGF Testbed Logger

Figure 1: Integrated Eagle/BDS-D system configuration.

Figure 1 shows the overall system configuration for the integrated Eagle/BDS-D system; the configuration is actually quite simple. All of the system's components attach to the SIMNET network. The SIU and Eagle model communicate with one another via Remote Procedure Calls (RPCs); all other nodes communicate with one another via SIMNET and IOP format PDUs ethernet network.

1.2.3 The IST Computer Generated Forces Testbed

Under the Intelligent Simulated Forces (ISF) contracts (N61339-89-C-0044 and N61339-92-C-0045), IST has been conducting research in the area of CGF systems. IST has developed a personal computer based CGF Testbed that connects to either a SIMNET or DIS network and provides an environment for testing CGF control algorithms.

The IST CGF Testbed is composed of two major subsystems: a Simulator and an Operator Interface (OI). Each subsystem is an IBM-compatible personal computer running IST-developed CGF software and is a node on the SIMNET network. The CGF Testbed generates entities that are fully functional in the SIMNET environment; it generates and accepts network packets (often called protocol data units or PDUs) allowing the CGF entities to interact with the entities on the network.

There is also a functionally equivalent version of the IST CGF Testbed that uses DIS format network packets instead of SIMNET. That version is not currently being used for the Integrated Eagle/BDS-D project. However, the purpose of task (9) of this project is to convert the Integrated Eagle/BDS-D IOP

to DIS compliance.

The CGF Simulator generates the CGF entities in the battlefield. The Simulator computes vehicle dynamics, tracks other simulation entities with remote entity approximation, performs Line of Sight (LOS) determinations between entities, and calculates combat results.

The Simulator also generates the behavior for the CGF entities, using its autonomous behavior modeling mechanisms. These behaviors are usually initiated by commands from the operator via the CGF OI. Once initiated, each behavior may consist of several steps and decision points which are performed automatically by the Simulator without operator intervention.

The CGF OI provides the operator with the ability to create and control simulated entities in the battlefield. It does not actually control the entities; rather, the operator's commands are communicated via the network in the form of non-PDU packets to a Simulator node which executes them. A CGF OI node acts as a front end to one or more Simulator nodes.

The CGF OI and Simulator software are designed to allow an CGF OI to control entities simulated on multiple Simulators and for a Simulator to simulate entities which are individually controlled by separate CGF OIs. During entity creation, the CGF OIs automatically balance the load of entities onto the available Simulators.

1.3 Nature of Engineering Change Proposal 1 (ECP1)

Work on the Integrated Eagle/BDS-D project has proceeded rapidly at IST since the project began. During that time, it has become clear that the basic concept of the project is sound. However, a number of shortcomings and limitations in the original project design and scope have been identified by IST, TRAC, and LANL.

ECP1 adds a set of new or modified tasks to the original project that are intended to address those limitations and increase the value and usefulness of the project. The new tasks were grouped into three broad categories. Those categories are:

1. Expanding the interface

These are tasks that will eliminate simulation and scenario restrictions imposed by Eagle and CGF Testbed software limitations. These tasks will increase the flexibility and applicability of the system.

2. Verification and validation

These tasks are designed to verify and validate the results of simulations executed in the integrated Eagle/BDS-D environment. These tasks will improve the credibility of results obtained from those simulation runs, especially for analysis purposes.

3. Experimental research and development

Tasks in this category attempt to take advantage of the unique aspects of the integrated Eagle/BDS-D simulation to advance the state-of-the-art in simulation and CGF technology.

When completed, the tasks described in this work plan will greatly increase the value of the Integrated Eagle/BDS-D system. It will simultaneously become more useful for combat development and analysis and more viable as an example of "seamless simulation", i.e. of how to integrate simulations of different granularity. Accomplishing these tasks will require approximately four person-years of full-time software engineers effort and approximately four person-years of part-time of part-time student research assistant effort.

1.4 Structure of this document

Section 2 describes the work to date on ECP1 tasks.

Section 3 provides a summary of the project and future plans.

2.0 Engineering Change Proposal 1 (ECP1)

This section details the work performed to date on ECP1 of this contract. The first section describes the work plan developed for the tasks in ECP1. The following four sections describes the four currently active tasks in ECP1: (2) Track Eagle units within CGF Testbed, (3) Base Eagle unit detection on LOS, (4) Platoon and battalion disaggregation, and (5) Include manned simulators.

2.1 ECP1 Work Plan

The first task of ECP1 was the preparation of a work plan for the tasks within ECP1. The work plan is detailed in report IST-CR-93-26, "Integrated Eagle/BDS-D Work Plan" which was prepared during the period covered by Interim Report 1 (IST-CR-93-31). This section summaries the work plan.

ECP1 consists of twenty tasks divided into three categories. The following table list the tasks by category:

Expanding the interface

(1) Add new vehicle types
(2) Track Eagle units within CGF Testbed
(3) Base Eagle unit detection on LOS
(4) Platoon and battalion disaggregation
(5) Include manned simulators
(6) Eliminate the fixed high-res area
(7) Make Eagle units visible on Stealth
(8) Control the Stealth from the CGF OI
(9) DIS compliance
(10) Perform full call for fire

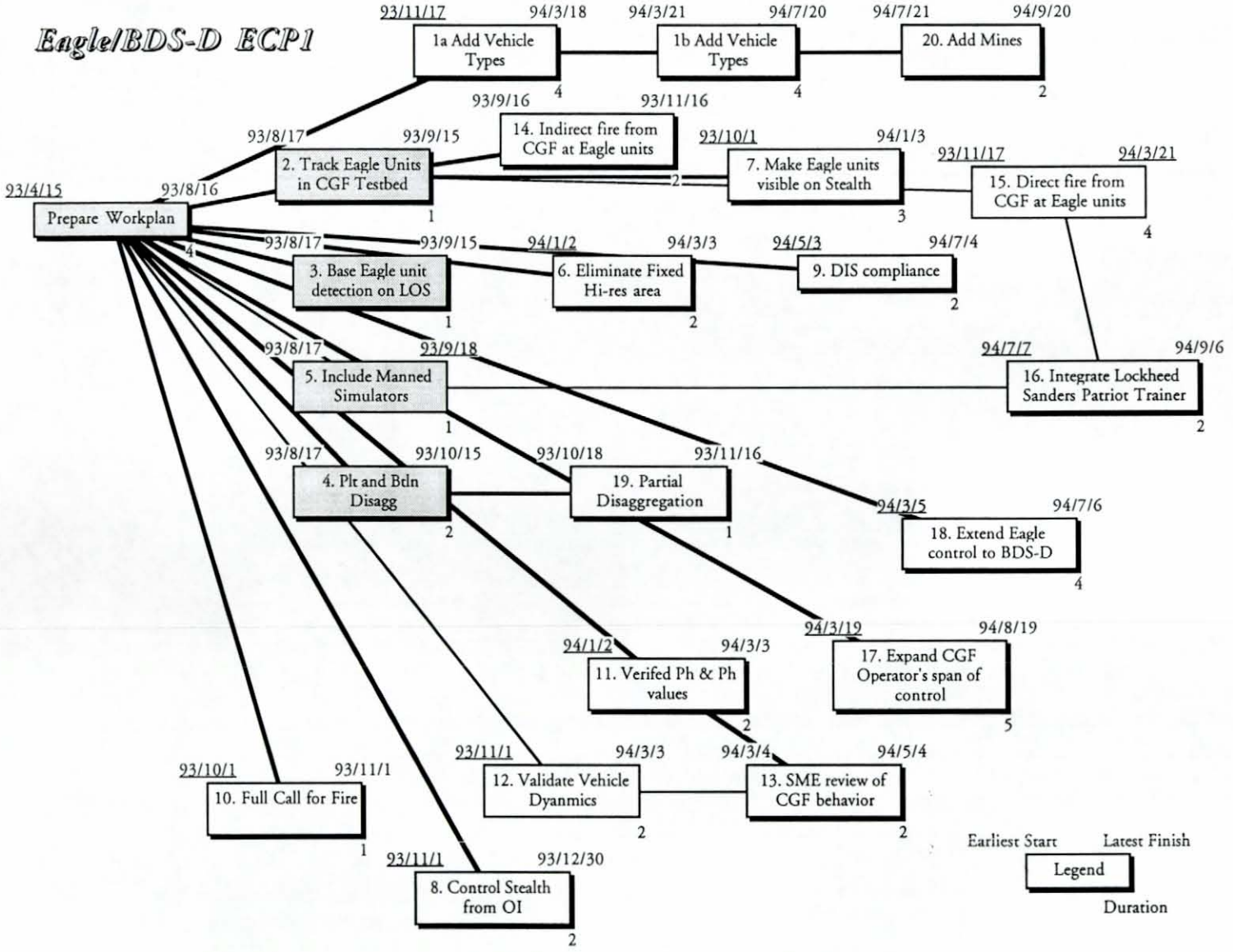
Verification and validation

(11) Use verified P_h and P_k values
(12) Validate vehicle dynamics model
(13) SME review of CGF behavior

Experimental research and development

(14) Indirect fire from CGF at Eagle units
(15) Direct fire from CGF at Eagle units
(16) Integrate the Lockheed Sanders Patriot
(17) Expand CGF operator's span of control
(18) Extend Eagle control to BDS-D
(19) Allow partial disaggregation
(20) Add mines

The work plan describes each task and, for each task, details the personnel resources (man-months of principal investigator, software engineer and research assistant time), expected start and end dates, inter-task dependencies among tasks, and subtasks within each task. The entire effort requires one full-time principal investigator, two full-time software engineers, and four part-time research assistants (graduate and under-graduate students). The plan calls for ECP1 to be completed by 15 September 1994. The chart on the next page shows a PERT chart of the ECP1 tasks. Shaded task boxes represent completed tasks. More detailed PERT and Gantt charts are available in the work plan.


Eagle/BDS-D ECP1
93/11/17
94/3/18
1a Add Vehicle Types
4
94/3/21
94/7/20
1b Add Vehicle Types
4
94/7/21
94/9/20
20. Add Mines
2
93/9/16
93/11/16
14. Indirect fire from CGF at Eagle units
2
93/8/17
93/9/15
2. Track Eagle Units in CGF Testbed
1
93/10/1
94/1/3
7. Make Eagle units visible on Stealth
3
93/11/17
94/3/21
15. Direct fire from CGF at Eagle units
4
93/4/15
93/8/16
Prepare Workplan
4
93/8/17
93/9/15
94/1/2
94/3/3
94/5/3
94/7/4
3. Base Eagle unit detection on LOS
1
6. Eliminate Fixed Hi-res area
2
9. DIS compliance
2
93/8/17
93/9/18
94/7/7
94/9/6
5. Include Manned Simulators
1
16. Integrate Lockheed Sanders Patriot Trainer
2
93/8/17
93/10/15
93/10/18
93/11/16
4. Plt and Bdn Disagg
2
19. Partial Disaggregation
1
94/3/5
94/7/6
18. Extend Eagle control to BDS-D
4
94/3/19
94/8/19
17. Expand CGF Operator's span of control
5
94/1/2
94/3/3
11. Verifed Ph & Ph values
2
93/11/1
94/3/3
94/3/4
94/5/4
12. Validate Vehicle Dyanmics
2
13. SME review of CGF behavior
2
93/10/1
93/11/1
10. Full Call for Fire
1
93/11/1
93/12/30
8. Control Stealth from OI
2
Earliest Start
Latest Finish
Legend
Duration

2.2 Task 2: Tracking Eagle Units in the CGF Testbed

Prior to this task, the IST CGF Testbed CGF OI displayed only the vehicles for which SIMNET Vehicle Appearance PDUs were being produced. Because Vehicle Appearance PDUs are not generated for Eagle units, the CGF operator has no information about the non-disaggregated Eagle units participating in the exercise. This tasks adds to the CGF OI a display of the non-disaggregated Eagle units and allows tracking the non-disaggregated Eagle units in the CGF system.

Section 2.2.1 describes an extension to this task which replaces the unit icon representing non-disaggregated Eagle units with individual vehicles in templated formations. These individual vehicles are visible in SIMNET as regular vehicles however they do not fire or react to fire; they simply move across the terrain in formation.

description

Expand the Interoperability Protocol (IOP) to include Unit Appearance PDUs (UAPDUs), which are the unit equivalent of Vehicle Appearance PDUs (VAPDUs). A UAPDU contains information identifying the unit, specifying its size and composition, and giving the location of its center of mass. UAPDUs are produced by the SIU and placed on the network. The Eagle CGF Manager will convert the UAPDUs to VAPDUs and places them on the network for CGF OI and Simulator nodes to handle.

The Eagle model periodically informs the SIU about the position, speed, heading, operational activity, relative strength, and effectiveness of the units in the exercise. The SIU builds, for each unit, a UAPDU and transmits it to the Eagle CGF Manager. The Eagle Manager stores the description of each unit so that multiple VAPDUs can be produced between reception of UAPDUs. VAPDUs must be produced no less frequently than one every 12 seconds in SIMNET; a vehicle or unit is considered to have exited the exercise if a VAPDU has not been received for 12 seconds. To ensure that VAPDUs are produced frequently enough to prevent units from "exiting" and "re–entering" SIMNET , the Eagle CGF Manager will dead reckon the units' positions between receipt of their UAPDUs and produce VAPDUs every 5-6 seconds.

subtasks

i. Define the contents of the UAPDU. **[Completed]**
ii. Modify Eagle CGF Manager to produce SIMNET VAPDUs from UAPDUs. **[Completed]**
iii. Modify the CGF OI to show Eagle units on its plan view display using the information in the Vehicle Appearance PDUs. Eagle units will be displayed using standard military symbols. **[Completed]**
iv. Adapt vehicle dead reckoning to units so that their displayed positions are updated between successive arrivals of UAPDUs. **[Completed]**
v. Coordinate necessary changes to SIU with LANL. **[Completed]**
vi. Define test scenario with TRAC and LANL. **[Completed]**
vii. Test functionality with test scenario and correct errors as needed. **[Completed]**

progress on subtasks

i. Define the contents of the UAPDU. **[Completed]**

The following Unit Appearance PDUs has been added to the Eagle IOP:

```
typedef struct  {
     unsigned long  vehicle;        /* Veh. Id to be used in VAPDU.    */
     EAGLE_LOCATION loc;            /* Location of the unit            */
     float          heading;        /* The units heading (radians)     */
     float          speed;          /* The unit speed (m/s)            */
     UCHAR          echelon_level; /* e.g. Company, Btln, Corps, etc. */
     UCHAR          echelon_type;  /*  e.g. Infantry, Armored, etc.   */
     UCHAR          alignment;     /* Side of unit                    */
     UCHAR          effectiveness; /* Current effectiveness           */
     unsigned short relative_strength;  /* Percent active vehicles     */
     unsigned short operation_activity; /* Operational Activity        */
     unsigned short remove;             /* Remove unit from exercise   */
     char           unit_name[MAX_UNIT_NAME_LENGTH];
} UNIT_APPEARANCE;
```

ii. Modify Eagle CGF Manager to produce SIMNET VAPDUs from UAPDUs **[Complete]**

The following functionality was added to the Eagle CGF Manager within the IST CGF Testbed simulator. A Unit Appearance Manager has been created in the CGF Testbed to manage UAPDUs. The Eagle CGF Manager was modified to receive UAPDUs and forward them to the Unit Appearance Manager which places them on a Unit_Appearance queue. Within the Unit Appearance Manager, a Finite State Machine (FSM) was built to traverse the Unit_Appearance queue every 5-6 seconds and produce a Vehicle Appearance PDU for each unit in the list with each unit being described as a SIMNET echelon (see below).

Vehicles (e.g. tanks) are distinguished from echelons (e.g. companies, battalions, etc.) in the Vehicle Appearance PDUs through their description in GUISE field which is a 32 bit field of type OBJECT_TYPE . The following is a description of the vehicle guises taken from BBN Report No. 7627 Appendix B page 149.

The 32 bit object type field which describes a vehicle/echelon is divided into the following fields, listed here from Most Significant Bit (MSB) to Least Significant Bit (LSB):

domain	3 bits	the value 1 denotes vehicle, the value 5 denotes an echelon
environment	3 bits	describes the environment : (e.g. Air, Ground, Space, Water)
class	3 bits	organizes vehicles within a particular environment into broad classes: (e.g. Rotary Wing, Armored Tracked, Armored Wheeled)
country	6 bits	describes the country that designed the vehicle (e.g., USA, USSR)
series	6 bits	identifies the level of echelon: (e.g. platoon, company, battalion)
model	6 bits	not used for echelons
function	5 bits	identifies the function of the vehicle/echelon (e.g., command post, armored cavalry, combat engineering)

For example, an Armored Battalion object type definition (in C) is:

```
#define BATTALION_ARMORED_ECHELON \
  (     OBJ_DOMAIN_ECHELON | \
        ECH_ENVIRONMENT_GROUND | \
        ECH_CLASS_SP_ARMORED_TRACKED | \
        ECH_COUNTRY_US | \
        ECH_KIND_BATTALION | \
        ECH_FUNC_MAIN_BATTLE_TANK )
```

The constants used in defining echelons are list in appendix A which is a partial listing of the file "objects.h". Note that echelon level (e.g. company) is referred to in SIMNET as echelon type and echelon type (e.g. infantry) is referred to in SIMNET as echelon function.

iii. Modify the CGF OI to show Eagle units on its plan view display using the information in the Vehicle Appearance PDUs. Eagle units will be displayed using standard military symbols.
The CGF OI has been modified to display on its plan view display standard military symbols for echelons described in Vehicle Appearance PDUs. The available SIMNET echelon levels and types are described in appendix A. **[Completed]**

iv. Adapt vehicle dead reckoning to units so that their displayed positions are updated between successive arrivals of UAPDUs. **[Completed]**
The CGF OI has been modified to dead reckon echelons (units) based on information in their most recent Vehicle Appearance PDUs.

v. Coordinate necessary changes to SIU with LANL. **[Completed]**
Completed at August 4-5 1993 Technical Interchange Meeting (TIM 6) in TRAC facilities in Ft. Leavenworth Kansas.

vi. Define test scenario with TRAC and LANL. **[Completed]**
IST, LANL, and TRAC personnel met at TRAC facilities in Ft. Leavenworth Kansas October 5-8 for a Techinical Interchange Meeting (TIM 7). At that meeting, a test scenario was devised and executed to test ECP tasks 2, 3, and 4. The trip report for this trip is included as appendix G. Briefly, the scenario involved:

1. Initialization of Eagle, SIU, and CGF system interconnection.
2. Eagle issues Unit Appearance PDUs have two blue and one red units.
3. Disaggregation of a Red unit.
4. Sending of Op Order for disaggregated Red unit.
5. Disaggregation of a Blue unit.
6. Sending Operator Intent messages from CGF OI.
7. Sending Indirect Fire Request from CGF OI.
8. Sending Frag Order from Eagle.
9. Request re-aggregation from Red unit's CGF OI.
10. Re-aggregation of Red unit.

vii. Test functionality with test scenario and correct errors as needed.
This task was tested at TIM 7 (see appendix G). During the course of the exercise, icons representing Eagle aggregate units were displayed on the CGF OI. Three minor software errors were detected and their corrections forwarded to TRAC the following week.

2.2.1 Task 2k: Detailed Unit Appearance

At TIM 7, Mr. Kent Pickett of TRAC requested an extension to task #2 to accommodate studies of helicopters as reconnaissance vehicles and studies of other sensor systems (e.g. JSTARS). In these studies, it is essential that the reconnaissance vehicles and sensor platforms be presented with numerous individual vehicles in the environment. This can be accomplished by replacing the icons representing aggregate units with vehicles in templated formations.

The goal of this extension to replace the single unit icon with vehicles in a templated pattern based on the unit's operational activity. Task #2 tracks and dead reckons Eagle aggregate units and issues single SIMNET Vehicle Appearance PDUs for each unit every 5 seconds as the units move around the battlefield. This extension will issue Vehicle Appearance PDUs for individual vehicles in a templated formation around the units' centers of mass. Note that only the unit is dead reckoned and that the individual vehicles are not simulated in any fashion; this will minimize the computational overhead of simulating many vehicles in the environment.

The following is the definition of the DETAILED_UNIT_APPEARANCE PDU (DUAPDU) that Eagle will send to the Eagle CGF Manger every time step.

```
/* Detailed Unit Appearance PDU. */
typedef struct
{
    DETAILED_UNIT_INFO   unit_info;     /* Specifications for disagg. */
    UCHAR           system_count;  /* Number of groups of entities in*/
                                   /* unit. */
    char            padding[3];    /* Forces data to 4 byte boundary. */
    char            data;          /* Array of 'systemCount' elements */
                                   /* of System Info.                 */
} DETAILED_UNIT_APPEARANCE_PDU;

typedef struct
{
    ULONG           start_vehicle_num; /* Vehicle num in VAPDU's      */
    EAGLE_LOCATION  loc;               /* Location of unit.           */
    float           heading;           /* The units heading (deg.)    */
    float           orientation;       /* Units orientation (deg.)    */
    float           speed;             /* The unit speed (m/s)        */
    UCHAR           echelon_level;     /* Level of echelon            */
    UCHAR           echelon_type;      /* Echelon type                */
    UCHAR           alignment;         /* Side of unit                */
    UCHAR           effectiveness;     /* Current effectiveness       */
    USHORT          operation_activity; /* Operational Activity       */
    USHORT          remove;            /* Remove unit from exercise   */
    char            unit_name[MAX_UNIT_NAME_LENGTH];
    char            system_count;      /* Number of systems in unit   */
    char            padding[2];        /* Force align. to 4 byte      */
                                       /*  boundary                   */
    char            data;              /* Array of 'sysCount' elements*/
                                       /*  of System Info.            */
} DETAILED_UNIT_INFO;
```

```
/* This is not a new structure; it exists already in loc_epro.h.  It is included here for clarity*/
typedef struct
{
    ULONG     entity_type;          /* DI, ABRAMS, BMP, etc.           */
    int       num_of_entities;      /* Num of this type in the unit    */
    char      padding[2];           /* Align to 4 byte boundary        */
} SYSTEM_INFO;
```

One of the issues we will resolve is how to minimize the frequency and duration of vehicles floating above or sinking below the terrain. Floating and sinking of vehicles will occur as the units move across the terrain because only the center of mass of the entire unit (not the individual vehicles) is being dead-reckoned as a straight line across the terrain. However, other nodes on the net will dead-reckon each vehicle using the vehicles' attitudes in their VAPDUs. When the attitudes of the unit and its vehicles differ, the vehicles float or sink. The approach that is being implemented determines the attitude (roll and pitch) of each vehicle from the normal of the polygon underlying each vehicle; the unit's heading determines the yaw component of the attitude. Each vehicle will move directly across the surface of its underlying polygon. This will minimize floating and sinking but not eliminate it because each vehicle will float or sink when it leaves a polygon. Experimentation will reveal an acceptable rate of recalculating the vehicle attitude relative to the underlying polygon. Note that the slower the unit is moving the less the problem.

To accomplish this task, two changes were made. First, the Eagle CGF Manager forwards DUAPDUs to the Unit Appearance Manager which places them on a DUAPDU list. When the DUAPDU is placed on the list, the positions of the individual vehicles relative to the unit's center of mass are calculated and stored with the DUAPDU. Second, a Detailed Unit Appearance FSM has been created in the Unit Appearance Manager. This FSM processes the "next" DUAPDU on the list, advances the "next" pointer, and reschedules itself with a delay proportional to the number of DUAPDUs on the list and the maximum time allowed to process the entire list.

Processing a DUAPDU involves generating VAPDUs for each vehicle in the DUAPDU. The location (x,y,z) of each vehicle is determined from the unit's center of mass and the vehicles' relative positions. The attitude of each vehicle is determined from the polygon underlying the vehicles. Each vehicle is identified by a Vehicle ID which consists of the Site and Host of the Unit Appearance Manager and a vehicle number. The SIU supplies the first vehicle number to be used in the Vehicle IDs; the Unit Appearance Manager assigns each vehicle a unique vehicle number beginning at the start_vehicle_num in the DUAPDU. The SIU is responsible for leaving sufficient "open" vehicle numbers between different units.

An early version of the Detailed Unit Appearance FSM produced VAPDUs for all the vehicles in all the units without a delay between units. This resulted in an uneven production of VAPDUs; e.g. a hundred VAPDUs might be produced in less than a second followed by 10 seconds with no VAPDUs being produced. To even the VAPDU production, a delay was inserted between units. Because the number of DUAPDUs on the list is uncertain and varies with time, the delay is recalculated each time a DUAPDU is processed as follows:

delay = seconds_to_process_list / number_of_DUAPDUs

"seconds_to_process_list" is a configuration file option and determines how many seconds the FSM takes to process the entire DUAPDU list. "number_of_DUAPDUs" is a variable which holds the current length of the DUAPDU list. With this approach, a single unit's VAPDUs are produced as a

group followed by a delay. For example, if "seconds_to_process_list" was set to 10 seconds and the current number_of_DUAPDUs was 5, there would be a 2 second delay between each unit.

Early testing of this task indicates that several hundred vehicles can be generated in response to Detailed Unit Appearance PDUs.

2.3 Task 3: Base Eagle Unit Detection on Line of Sight

Eagle units are typically disaggregated out of line of sight of one another. When the vehicles of a disaggregated unit move within line of sight of one another in the virtual environment, they detect, recognize, and identify one another using a realistic sighting model for the 3-dimensional virtual environment built into the Simulator. The goal of this task is to provide to the Eagle model realistic unit sighting reports based on vehicle sightings in the virtual environment.

description

CGF vehicles perform line of sight (LOS) checks in the SIMNET battlefield and determine sightings based on a realistic sighting model. The sightings are reported via a network PDU sent from the Simulator to an OI. The sighting report could be captured by the SIU and converted into an Eagle detection.

This task is to modify the CGF Testbed's sighting reports and sighting model to forms in which they are useful to the Eagle system for unit detection.

subtasks

i. Determine with LANL if existing Sighting Report PDU is acceptable. If not, modify Eagle CGF Manager to produce Sighting Report within IOP. [**Completed**]
ii. Coordinate necessary changes to SIU with LANL. [**Completed**]
iii. Obtain sighting models from TRAC and AMSAA.. [**Completed**]
iv. Implement new sighting model or modify existing sighting model as necessary. [**Completed**]
v. Define test scenario with TRAC and LANL. [**Completed**]
vi. Test functionality with test scenario and correct errors as needed. [**Completed**]

progress on subtasks

i. Determine with LANL if existing Sighting Report PDU is acceptable. If not, modify Eagle CGF Manager to produce Sighting Report within IOP. [**Completed**]
LANL has been supplied with a detailed description of the CGF Testbed's Sighting Report PDU. The SIU was modified to accept and process these PDUs.

ii. Coordinate necessary changes to SIU with LANL. [**Completed**]
Initiated at August 4-5 1993 Technical Interchange Meeting (TIM 6) in TRAC facilities in Ft. Leavenworth Kansas.

iii. Obtain sighting models from TRAC and AMSAA.. [**Completed**]
Sighting model obtained from AMSAA.

iv. Implement new sighting model or modify existing sighting model as necessary. [**Completed**]
Existing sighting model judged adequate for this task.

v. Define test scenario with TRAC and LANL. [**Completed**]
IST, LANL, and TRAC personnel met at TRAC facilities in Ft. Leavenworth Kansas October 5-8 for a Techinical Interchange Meeting (TIM 7). At that meeting, a test scenario was devised and executed to test this task. The trip report for this trip is included as appendix G.

vii. Test functionality with test scenario and correct errors as needed. [**Completed**]
This task was tested without errors. The SIU processed CGF Testbed Sighting Reports generated by disaggregated vehicles and reported those sightings to the Eagle model. The Eagle model then adjusted its representation of unit sightings to reflect the sightings between vehicles.

2.4 Task 4: Platoon and Battalion Disaggregation

In the first phase of the Integrated Eagle/BDS-D project only companies were disaggregated. This task introduces the ability to disaggregate platoons and battalion size units. Platoon sized disaggregations are necessary for task (19) Partial Disaggregation. Because the smallest maneuver units in the Eagle model are battalions and companies, battalion size disaggregations will complete the ability to disaggregate the Eagle maneuver units.

description

Allow disaggregation and aggregation of platoon and battalion sized units, in addition to the current company sized units.

subtasks

i. Expand the Disaggregation Formation Template structures to accommodate battalions. **[Completed]**
ii. Redesign and implement Disaggregation Process: **[Completed]**
   a. Design generic communication/command entity for Battalion, Company, and Platoon level.
   b. Design and implement OI Supervisor Manager to coordinate requests for OIs to control entities in the Disaggregation Requests.
   c. Design and implement recursive disaggregation process (coordinate with task 5).
   d. Design and implement the changes to the Disaggregation Request data structures to accommodate battalion and platoon sized units.
   e. Design and implement the changes to the Disaggregation Response data structures to accommodate battalion and platoon sized units.
iii. Coordinate necessary changes to SIU with LANL. **[Completed]**
vii. Define test scenario with TRAC and LANL. **[Completed]**
viii. Test functionality with test scenario and correct errors as needed. **[Completed]**

progress on subtasks

i. Expand the Disaggregation Formation Template structures to accommodate battalions. **[Completed]**

The unit composition definitions in the first phase of the project were specific to company sized units. Those data structures and definitions were not suitable for expansion to battalion sized units. Although adequate for company sized units, they could not be expanded to accommodate battalion size units due to memory constraints. Additionally, the amount of redundant data caused difficulties in maintenance. To solve these problems, two grammars were developed to describe the composition and formation of units (see appendix B for the grammars). These grammars allow unit compositions and unit formations to be described concisely.

Here is an example of how the composition and formation grammars are used to define a unit; in this case the CavTroop company (appendix C and D contains the complete definitions). The following statement defines the composition of a CavTroop company:

```
UNIT CavTroop :=
{
   Type          := Company;
   SubUnits      := HQSec, Plt_1, Plt_2, Plt_3;
   SubUnitMap    := CavTroopSubUnitMap;
   Vehicles      := ;
   VehicleMap    := ;
}
```

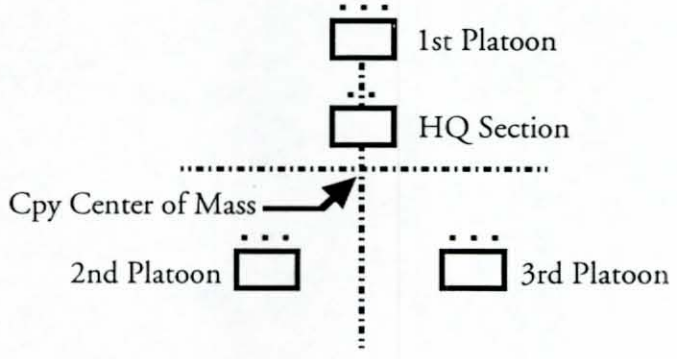
The SubUnits field specifies that a CavTroop company contains four subunits. The SubUnitMap defines the positions of the subunits around the company center of mass for each of the five formations (Assembly, Wedge, Vee, Line, Column). The CavTroopSubUnitMap is defined as:

```
MAP CavTroopSubUnitMap :=
{
    Assembly        => CavTroopSubUnitAssembly,
    Wedge           => CavTroopSubUnitWedge,
    Vee             => CavTroopSubUnitVee,
    Line            => CavTroopSubUnitLine,
    Column          => CavTroopSubUnitColumn
};
```

Each of the formations is described by a separate definition. For example, the formation of the subunits (platoons) with the CavTroop company when a "wedge" formation is given by the CavTroopSubUnitWedge definition:

```
CavTroopSubUnitWedge := { (0 30) (0 75) (-200 -75) (200 -75) };
```

The CavTroopSubUnitWedge specifies the set of points that defines the position of each of the sub-units with respect the company center. The CavTroopSubUnitWedge mapping would look like:



The composition grammar allows us to define command vehicles at each echelon. The command vehicles would be specified in the Vehicles field and their positions relative to the company center of mass would be described by an entry in the "VehicleMap" field. For example, the Headquarters section can be replaced by its individual vehicles as follows:

```
UNIT CavTroop :=
{
    Type            := Company;
    SubUnits        := Plt_1, Plt_2, Plt_3;
    SubUnitMap      := CavTroopSubUnitMap;
    Vehicles        := M1, M2;
    VehicleMap      := HQSecVehMap;
}
```

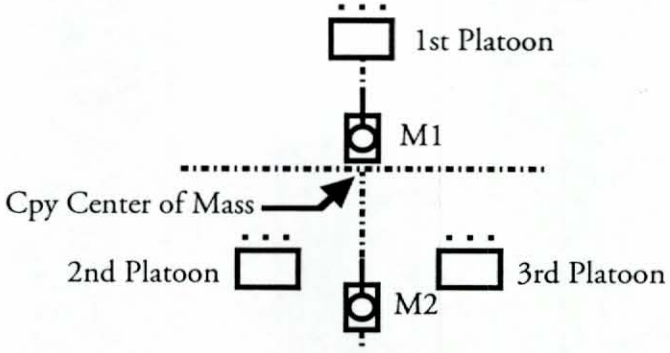
With this statement in the composition grammar, the CavTroop company is defined with two vehicles at the company level and three platoon sized subunits. With this definition of the CavTroop, the CavTroopSubUnitMap would be:

CavTroopSubUnitWedge := { (0 75) (-200 -75) (200 -75) };

and the HQSecVehMap would be:

HQSecVehMap := { (0 15) (0 -85) }.

The CavTroopSubUnitWedge mapping might look like:



ii. Redesign and implement Disaggregation Process: **[Completed]**

For this task, the disaggregation process was extended to accommodate disaggregations of platoons and battalions. Analysis of the requirements for battalion, company, and platoon level disaggregations led to three significant changes. First, a generic Communication/Command (C^2) entity for the battalion, company, and platoon levels was developed rather than creating a new battalion C^2 entity and a new platoon C^2 entity. Second, a new manager entity, the OI Supervisor Manager, was developed to coordinate requests for OIs from all echelons in a disaggregation request. Third, based on common features of the disaggregation process at a the battalion, company, and platoon levels, the disaggregation process was redesigned as a simple, general process suitable at all three levels. This approach is also suitable for echelons above battalion within the limitations of the vehicle level simulation to support the number of vehicles involved. This disaggregation process is termed the "distributed disaggregation" process because the functionality of subunit creation and vehicle placement is distributed throughout the C^2 entity hierarchy rather than centralized to one entity as was the case in the original disaggregation process.

a. Design generic communication/command entity for Battalion, Company, and Platoon level.

The composition grammar (described above) provides a mechanism for describing the composition of the echelons in a consistent fashion. Based on this composition facility, a generic Communication/Command (C^2) entity for the battalion, company, and platoon levels was designed and implemented. This generic C^2 entity is responsible for, first, instantiating each of its command vehicles, second, creating C^2 entities for of its subunits, and, third, requesting each of its subunit C^2 entities to disaggregate their set of vehicles and

subunits.

b. Design and implement OI Supervisor Manager to coordinate requests for OIs to control entities in the Disaggregation Requests.

The new composition structure and the new distributed disaggregation process has also prompted a change to the way in which OI Supervisors are handled. Because each of the generic C^2 entities may have command vehicles associated with it, all the C^2 entities (not just the lowest level platoon C^2 entities) need to be provided with access to OI Supervisors to allow the command vehicles can be instantiated.

An OI Supervisor Manager has been designed and implemented that is responsible for keeping track of all OI Supervisors that are available to the C^2 entities. When individual vehicles are being instantiated, each C^2 entity asks the OI Supervisor Manager for an OI Supervisor to coordinate the instantiation process. The OI Supervisor Manager determines which OI Supervisor best fits the current request (a greedy algorithm is used) and returns its identifier to the requesting C^2 entity. The C^2 entity then communicates directly to the OI Supervisor to create and activate individual vehicles.

c. Design and implement disaggregation process (coordinate with task 5).

Based on the composition grammar for echelons, the design of the generic C^2 entity, and the design of the OI Supervisor Manager, the disaggregation process has been redesigned and implemented as a simpler, more general, process that uses a common algorithm at all echelon levels rather than specific routines at each level of the echelon hierarchy. This new design allows a combination of battalion, company, and platoon level disaggregations to occur together without interference. It also allows the instantiation of command vehicles and subunits of varying composition to occur throughout the echelon hierarchy.

d. Design and implement the changes to the Disaggregation Request data structures to accommodate battalion and platoon sized units.

The Disaggregation Request data structure has been extended to allow battalion and platoon disaggregations. The changes consist of expanding the list of allowed echelons to include battalion and platoon sized units.

e. Design and implement the changes to the Disaggregation Response data structures to accommodate battalion and platoon sized units.

The Disaggregation Response reports to the SIU the vehicle ids of all vehicles created in a disaggregation. The Disaggregation Response PDU can not hold all the vehicle ids in a battalion. The disaggregation process has been changed to return a series of Disaggregation Response PDUs for battalion disaggregations. Sequence and count fields have been added to the Disaggregation Response PDU to specify the number of Response PDUs being generated for a single disaggregation and the order of the PDUs.

iii. Coordinate necessary changes to SIU with LANL. **[Completed]**
Initiated at August 4-5 1993 Technical Interchange Meeting in TRAC facilities in Ft. Leavenworth Kansas.

iv. Define test scenario with TRAC and LANL. **[Completed]**
Completed at October 5-8 Technical Interchange Meeting (TIM 7).

vii. Test functionality with test scenario and correct errors as needed. **[Completed]**
IST, LANL, and TRAC personnel met at TRAC facilities in Ft. Leavenworth, Kansas October 5-8 for a Technical Interchange Meeting (TIM 7). At that meeting, the ability to disaggregate battalions and platoons was demonstrated through the use of logged Disaggregation Request for platoons and small, notional battalions. The use of logged Disaggregation Requests was necessary because TRAC had insufficient computers to disaggregate a full battalion and the Eagle model does not currently represent platoons.

2.5 Task 5: Include Manned Simulators

For Integrated Eagle/BDS-D POC1, all vehicles within a disaggregated unit were controlled by the IST CGF Testbed. This task allows manned simulators to be part of disaggregated units. With this task complete, a disaggregated unit may consist entirely of CGF vehicles, manned simulators, or a mixture of CGF vehicles and manned simulators.

description

Extend the IOP Disaggregation request PDU and the CGF Testbed's entity creation process to include manned simulators, such as M1 and M2 simulators, in the disaggregation and aggregation operations.

subtasks

i. Capture Activation Request (AR) PDUs. [**Completed**]
ii. Reverse engineer AR and VR PDUs. [**Completed**]
iii. Design and implement Manned Simulator Manager. [**Completed**]
iv. Design and implement Manned Simulator Configuration File (eagle.man). [**Completed**]
v. Change disaggregation and aggregation processes to use manned simulators. [**Completed**]
   • Design and implement recursive disaggregation process (coordinate with task (4)).
vi. Coordinate necessary changes to SIU with LANL. [**Completed**]
vii. Define test scenario with TRAC and LANL. [**Completed**]
viii. Test functionality with test scenario and correct errors as needed. [**Completed**]

progress on subtasks

i. Capture Activation Request (AR) PDUs. [**Completed**]
In SIMNET one simulator may activate another simulator by sending the second simulator an Activation Request (AR) PDU. To verify the contents of a valid AR PDU, an AR PDU generated by the MCC system to activate an M1 simulator was logged.

ii. Reverse engineer AR PDU. [**Completed**]
From the logged AR PDU, the correct values of the various fields (e.g. Fuel levels, Ammunition levels, subsystem status, etc.) of an M1 simulator Activation Request were determined. The CGF Simulator was modified to produce M1 Activation Request PDUs. Because the Simulator could produce only Datagram PDUs and the AR PDU is a Request Association PDU, this involved adding to the Simulator the capability to produce Request Association PDUs.

iii. Design and implement Manned Simulator Manager. [**Completed**]
To manage the activation of multiple manned simulators and multiple types of manned simulators, a Manned Simulator Manager has been added to the Eagle CGF testbed. The Manned Simulator Manager is responsible for maintaining a list of manned simulators, sending AR PDUs to the manned simulators in response to requests for manned simulators from the disaggregation process, and for sending DEACTIVATION REQUEST PDUs to release manned simulators during the aggregation process.

iv. Design and implement Manned Simulator Configuration File (eagle.man). [**Completed**]
A Manned Simulator Configuration File (eagle.man) has been implemented to store the information about how each manned simulator is to be used during an exercise.

The following is an example configuration file:

NUMBER_OF_SIMULATORS	1
SITE	17
HOST	15
VEHICLE_ID	0
ALIGNMENT	BLUEFORCE
VEHICLE_TYPE	ABRAMS
UNIT_NAME	TF_1_24
BATTALION_TYPE	BTLN_ARMORED
COMPANY_TYPE	ANY
PLATOON_TYPE	ANY
VEHICLE_NUMBER	10

This configuration file specifies that there is one manned simulator (vehicle id = (17 15 0)) and that it should be disaggregated as an M1 in the Blue force, in an armored battalion named "TF_1_24", in any company, in any platoon, and as the tenth M1 in the battalion. The keyword "ANY" may be used in the unit type fields or the unit name field to avoid specifying the unit precisely. The Manned Simulator Manager will respond to requests for manned simulators by matching the requesting unit with the information in the configuration file and allocating only those simulators that should be activated for the requesting unit.

v. Change disaggregation and aggregation processes to use manned simulators. **[Completed]**
In conjunction with task (4), the disaggregation process was modified as follows. Prior to creating a vehicle in the CGF system, the Manned Simulator Manager is queried to see if this vehicle should be a manned simulator. This is accomplished by sending an ACQUIRE_SIMULATOR message to the Manned Simulator Manager which specifies the type of vehicle needed, the details of the requesting unit, and where within the unit the vehicle is to be placed. The Manned Simulator Manager determines if a simulator is available that satisfies the request (e.g. Same alignment, vehicle, echelon etc.). If the request can be satisfied an AR PDU will be sent to the simulator which places it on the battlefield. If the request cannot be satisfied, a message is sent back stating why the request could not be met (e.g. No simulators available, None meet selection criteria, etc.).

To deactivate manned simulators during the aggregation process a RELEASE_SIMULATOR message is sent to the Manned Simulator Manager by the aggregation process. The Manned Simulator Manager upon receipt of the RELEASE_SIMULATOR message simply deactivates all simulators associated with the unit being aggregated. The simulators should be released by sending them a DEACTIVATION REQUEST PDU. However, the M1 simulator at IST doesn't appear to be SIMNET 6.6.1 compliant; it fails to recognize DEACTIVATION REQUEST PDUs. As a work-around, the manned simulators will be deactivated by moving them to (0,0) on the terrain by sending them a second AR PDU with location (0,0).

vi. Coordinate necessary changes to SIU with LANL. **[Completed]**
Initiated at August 4-5 1993 Technical Interchange Meeting (TIM 6) in TRAC facilities in Ft. Leavenworth Kansas.

vii. Define test scenario with TRAC and LANL. **[Completed]**
The test scenario for this task was the test scenario devised at the TIM 7. Because TRAC does not have a manned simulator on site, the test was performed at IST the week following TIM 7.

viii. Test functionality with test scenario and correct errors as needed. **[Completed]**

Disaggregation of the Blue unit included the M1 manned simulator as the company commander vehicle. When the company was disaggregated, the M1 simulator was positioned in the correct location. The M1 crew was able to see the other members of the company through the M1's viewports, to drive the M1 along with the company, and to participate in direct combat with a disaggregated OPFOR company.

2.6 Task 1: Add new vehicle types

description

Expand the set of vehicle/entity types that are simulated in the IST CGF Testbed and are available for disaggregation. Adding these new vehicle types will permit a wider range of scenarios in the integrated Eagle/BDS-D environment.

subtasks

i. The vehicles types to be added are: **[In Progress]**
   a. Rotary wing aircraft: AH-64 Apache, Mi-28 Havoc.
   b. Fixed wing aircraft: A-10 Thunderbolt II, Su-25 Frogfoot.
   c. Ground vehicles: M109 155mm SPA, Patriot launcher, Patriot radar vehicle, Patriot control vehicle, ZSU-23-4, ADATS, BTR APC, M113 APC.

ii. These vehicle types all exist as SIMNET 6.6.1 models, i.e. they can be shown visually using SIMNET-compatible Stealth devices. To add them to the IST CGF Testbed, enhancements are needed to the following functionality areas:
   a. wheeled vehicle dynamics and aircraft flight dynamics
   b. tactical maneuvers, e.g. ground attack runs
   c. new weapons, including Maverick, Hellfire, Stinger, Patriot, and Avenger 30mm.

progress on subtasks

i. The vehicles types to be added are: **[In Progress]**
   c. Ground vehicles: M109 155mm SPA, Patriot launcher, Patriot radar vehicle, Patriot control vehicle, ZSU-23-4, ADATS, BTR APC, M113 APC.
   In conjunction with task #14 - Indirect Fire from CGF at Eagle Units, the M109 155mm SPA is being added to the IST CGF Testbed. The behavior of the M109 is based upon information obtained from Maj. Segres, an artillery subject matter expert located at TRAC facilities in Ft. Leavenworth (see appendix G). See section 2.10 Task 14: Permit Indirect Fire from CGF Entities at Eagle Units for a more detailed description of the M109.

2.7 Task 7: Make Eagle units visible on the Stealth

description

As shown in Figure 1, the integrated Eagle/BDS-D system includes a Stealth display. Currently, that Stealth will display only individual vehicles that have been created as the result of a disaggregation. Modify that Stealth display to also show icons for aggregated Eagle units. Those units can be identified using the Unit Appearance PDUs defined for task (2).

As described earlier in task #2k(the extension to task #2 - Track Eagle Units in CGF Testbed), Detailed Unit Appearance PDUs can be sent from Eagle to the Eagle CGF Manager which produces Vehicle Appearance PDUs (VAPDUs) for the vehicles in the unit. These VAPDUs position the vehicles in the unit in a templated formation based on the operational activity of the unit. Thus, an aggregate Eagle unit appears as a set of individual vehicles moving in formation. This is termed a "detailed" unit appearance to distinguish it from displaying a single icon for the unit.

This task consists of two parts. The first is associated with task #2k and the second is the definition of the icon based unit display.

subtasks

i. In conjunction with task #2k, implement detailed unit appearance process. **[Completed]**
   a. Modify Eagle CGF Manger to produce VAPDUs from Unit Appearance PDUs.
   b. Coordinate necessary changes to SIU with LANL.
   c. Define test scenario with TRAC and LANL.
   d. Test functionality with test scenario and correct errors as needed.

ii. Implement icon based Stealth display. **[In Progress]**
   a. Define Unit Icons (size, shape, and markings) for display on Stealth.
   b. Learn visual modeling tools.
   c. Implement new "vehicles" (with Unit Icons) in Stealth vehicle tables for each unit type.
   d. Coordinate necessary changes to SIU with LANL.
   e. Define test scenario with TRAC and LANL.
   f. Test functionality with test scenario and correct errors as needed.

progress on subtasks

i. In conjunction with task #2k, implement detailed unit appearance process. **[Completed]**
   a. Modify Eagle CGF Manger to produce VAPDUs from Unit Appearance PDUs.
   Completed as part of task 2k.

   b. Coordinate necessary changes to SIU with LANL.
   The definitions of the Detailed Unit Appearance PDU has been agreed upon with LANL.

   c. Define test scenario with TRAC and LANL.
   A test scenario has been defined which incorporates several Detailed Unit Appearance requests from Eagle. The test scenario has been installed at IST.

   d. Test functionality with test scenario and correct errors as needed.
   This functionality will be included in next test and/or demonstration of project in December 1993 or January 1994.

ii. Implement icon based Stealth display. **[In Progress]**

a. Learn visual modeling tools.
A three day class on visual modeling tools for SIMNET was scheduled for December 7-9, 1993. Curt Lisle of the IST Visual Systems Lab taught the class. The purpose of the class was to train several people in the IST CGF Lab in the tools to create visual models for the SIMNET Stealth.

b. Define Unit Icons (size, shape, and markings) for display on Stealth.
Standard military markings will be used. The maximum size of the icons will be determined during task ii.a. Learn visual modeling tools.

c. Implement new "vehicles" (with Unit Icons) in Stealth vehicle tables for each unit type.
To be performed after task ii.a Learn visual modeling tools.

d. Coordinate necessary changes to SIU with LANL.
No changes needed.

e. Define test scenario with TRAC and LANL.
To be done in December 1993 or January 1994 Technical Interchange Meeting.

f. Test functionality with test scenario and correct errors as needed.
To be done in December 1993 or January 1994 Technical Interchange Meeting.

2.8 Task 8: Control the Stealth from the CGF OI

description

Implement control of the Eagle/BDS-D system's Stealth from the CGF OI. Ensure that the Stealth can handle standard Stealth control packets, which allows the operator to order the Stealth to attach, tether, and move. A function of particular usefulness is for the CGF operator to be able to teleport the Stealth point of view to that of a CGF entity that the operator selects on the CGF OI plan view display.

subtasks

i. Capture attach, tether, move, and teleport Stealth PDUs. **[Completed]**
ii. Reverse engineer Stealth PDUs, if necessary. **[Not Necessary]**
iii. Implement Stealth control facility in CGF OI. **[In Progress]**

progress on subtasks

i. Capture attach, tether, move, and teleport Stealth PDUs. **[Completed]**
These PDUs have been isolated and defined.

ii. Reverse engineer Stealth PDUs, if necessary. **[Not Necessary]**
This subtask is not necessary.

iii. Implement Stealth control facility in CGF OI. **[In Progress]**
A "Stealth" option has been added to the CGF OI menu under the "Master" option. When the "Stealth" mode is active, the arrow and page movement keys will control the Stealth by sending Stealth PDUs to the Stealth. The arrow keys will move the Stealth left, right, forward, and backward while the page up/down keys will increase/decrease the Stealth's altitude.

Additional menu options under the "Stealth" option are being added to attach, tether, and teleport the Stealth.

2.9 Task 10: Perform full call for fire

description

The purpose of this task is to enhance the existing, simple Call for Fire (Indirect Fire Request) to a request for indirect fire that supplies the information that Eagle requires for planning indirect fire. For this task the human CGF operator is acting as a Forward Observer (FO) and requesting indirect fire from artillery simulated in the Eagle model.

subtasks

i. Define (with TRAC and LANL) the call for fire process. **[Completed]**
   • Define new IOP PDUs for call for fire.
ii. Implement the CGF component of the call for fire process. **[Completed]**
iii. Coordinate necessary changes to SIU with LANL. **[Completed]**
iv. Define test scenario with TRAC and LANL. **[Not Begun]**
v. Test functionality with test scenario and correct errors as needed. **[Not Begun]**

progress on subtasks

i. Define (with TRAC and LANL) the call for fire process. **[Completed]**

The Call for Fire process was defined at the August 4-5, 1993 Technical Interchange Meeting (TIM 6) and is based on the existing Indirect Fire Request process but adds the following information:

Desired effect of fire : from enumeration (suppress, neutralize, destroy)
Target echelon level : from enumeration (platoon, company, battalion, battery)
Target echelon type : from enumeration (tanks, apc, infantry, helicopter, command post, artillery, mortar, air defense)
Terrain cover : from enumeration (open, covered)
Vegetation : from enumeration (light, medium, dense)

IST is responsible for generating, from human input at a CGF OI, Call for Fire PDUs. LANL is responsible for having the SIU forward the Call for Fire PDUs to the Eagle model. TRAC is responsible for altering the Eagle model to use the Call for Fire PDUs in its target value analysis and its allocation of indirect fire.

• Define new IOP PDUs for call for fire.

The following Call for Indirect Fire PDU has been added to the Eagle IOP:

```
typedef struct
{
    UNIT_ID         requesting_unit;    /* Unit requesting fire       */
    EAGLE_LOCATION  target_loc;         /* Fire at location           */
    UCHAR           effect;             /* Desired effect of fire     */
    UCHAR           target_ech_level;   /* Target echelon level       */
                                        /*  (e.g. Company, Btln)      */
    UCHAR           target_ech_type;    /* Target echelon type        */
                                        /*  (e.g. Armor, Infantry)    */
    UCHAR           cover;              /* Terrain cover              */
    UCHAR           vegetation;         /* Vegetation around target   */
} CALL_FOR_FIRE;
```

ii. Implement the CGF component of the call for fire process. **[Completed]**
New menus have been added to the CGF OI to allow the operator to create Call for Fire requests. These requests are passed to the Eagle CGF Manager which converts them to Eagle IOP Call for Fire PDUs and transmits them to the SIU. This functionality has been tested at IST and at LANL.

iii. Coordinate necessary changes to SIU with LANL. **[Completed]**
Completed.

iv. Define test scenario with TRAC and LANL. **[Not Begun]**
This functionality will be included in next test and/or demonstration of project in December 1993 or January 1994.

v. Test functionality with test scenario and correct errors as needed. **[Not Begun]**
This functionality will be included in next test and/or demonstration of project in December 1993 or January 1994.

2.10 Task 14: Permit indirect fire from CGF entities at Eagle units

description

Generate indirect fire from CGF entities, targeted at the locations of aggregated Eagle units. The locations of the Eagle units are known to the CGF Testbed from the Unit Appearance PDUs described earlier in task (2).

Indirect fire missions will be ordered either by the indirect fire control algorithms in Eagle or by the CGF operator using the CGF Testbed's OI. In the former case, the ordered indirect fire mission will be relayed to the CGF Testbed via the network and the SIU. In both cases, the CGF Testbed will generate the indirect fire from CGF entities with indirect fire capabilities, such as M109 vehicles. BDS-D Indirect Fire PDUs will be produced to affect entities in the virtual environment. In addition, the indirect fire will be transferred via the network (via IOP Indirect Fire Volley PDUs) to Eagle for resolution by Eagle against the target Eagle unit.

subtasks

1. For operator ordered indirect fire missions: [**Completed**]
   • Design and implement fire mission command structure in CGF OI.
2. For Eagle ordered indirect fire missions: [**Not Begun**]
   • Design and implement fire mission requests in IOP.
3. Design and implement fire mission request message passing and vehicle task assignment in CGF unit command structure. [**Completed**]
4. Design and implement indirect fire production from CGF Testbed vehicles. [**In Progress**]
5. Design IOP Indirect Fire Volley PDU. [**In Progress**]
6. Design and implement algorithm to consolidate BDS-D Indirect Fire PDUs into IOP Indirect Fire Volley PDU. [**In Progress**]
7. Coordinate necessary changes to SIU with LANL. [**In Progress**]
8. Define test scenario with TRAC and LANL. [**Not Begun**]
9. Test functionality with test scenario and correct errors as needed. [**Not Begun**]

progress on subtasks

1. For operator ordered indirect fire missions: [**Completed**]
   • Design and implement fire mission command structure in CGF OI.

   A new menu option is being implemented to specify a "Battery Fire Mission" on the CGF OI. The human operator will specify either a "sheath" or "point" target. For the sheath target, the width, depth, orientation, and the center of the sheath are entered. For the point target, the location of the target is entered. For both target types, the number of volleys and the munition type are entered. The "Battery Fire Mission" message is sent from the CGF OI to its OI Supervisor which sends it to the battery C^2 node. The Battery Fire Mission is defined as:

```
typedef struct
{
    UNIT_ID           unit_id;          /* Requesting unit_id          */
    OBJECT_TYPE       munition;         /* Type of munition            */
    EAGLE_LOCATION    target_loc;       /* Target loc for Battery Fire*/
    USHORT            number_of_volleys;/* No of volleys to be fired */
    UCHAR             point_or_sheath; /* Point or sheath fire        */
    USHORT            width;            /* Width of the sheath         */
    USHORT            depth;            /* Depth of the sheath         */
    USHORT            orientation;      /* Orientation of the sheath   */
} BATTERY_FIRE_MISSION;
```

2. For Eagle ordered indirect fire missions: [**Not Begun**]
• Design and implement fire mission requests in IOP.
This subtask is awaits definition by TRAC. Initially an operator ordered indirect fire missions will be implemented. When TRAC defines how to separate artillery fire missions among non-disaggregated and disaggregated artillery battalions, the mechanism will be in place on the IST CGF side to accept and implement the missions.

3. Design and implement fire mission request message passing and vehicle task assignment in CGF unit command structure. [**Completed**]
Based on discussions with Maj. Segres, an artillery subject matter expert, at TRAC during TIM 6 and TIM 7, the following approach has been taken. A U.S. Army artillery battery consisting of two artillery platoons each with four M109 Self Propelled Artillery vehicles has been defined within the CGF Testbed. During disaggregation, a battery C^2 node and two platoon C^2 nodes are created. The activity of the Fire Direction Officer (FDO) is contained within the battery C^2 node. The battery C^2 node receives the Battery Fire Mission (see above), determines the target locations of the individual guns, and sends "Platoon Fire Mission" messages to its platoon C^2 nodes. The Platoon Fire Mission is defined as:

```
typedef struct
{
    USHORT              mission_id;           /* mission identifier     */
    OBJECT_TYPE         munition;             /* munition to fire       */
    USHORT              number_of_volleys;    /* volleys in mission     */
    VOLLEY_VECTOR       volley_advance_vector;/* changes target loc     */
    USHORT              number_of_gun_locations;  /*number of guns (n) */
    data;                                     /* array of GUN_LOCATIONs */
} PLATOON_FIRE_MISSION;

typedef struct
{
    float x;
    float y;
} GUN_LOCATION;
```

The platoon C^2 node creates and sends "Gun Fire Mission" messages to each of its active M109s. The Gun Fire Mission is defined as:

```
typedef struct
{
    VEHICLE_ID      gun_id;                 /* the gun firing         */
    OBJECT_TYPE     munition;               /* the munition to fire   */
    USHORT          number_of_volleys;      /* how many times         */
    GUN_LOCATION    gun_target_initial_loc; /* initial target loc     */
    VOLLEY_VECTOR   volley_advance_vector;  /* change targ loc.       */
} GUN_FIRE_MISSION;
```

Each M109 fires its mission reporting each round fired to it platoon C² node in a "Round Complete" message. The Round Complete message is defined as:

```
typedef struct
{
    VEHICLE_ID      gun_id;             /* the gun firing              */
    GUN_LOCATION    detonation_loc;     /* location of detonation      */
    USHORT          round_complete;     /* which rd fired e.g. 4 of 8 */
    USHORT          round_sequence;     /* rds in volley e.g. 8        */
    UCHAR           gun_status;         /* active or destroyed         */
}ROUND_COMPLETE_MESSAGE;
```

The platoon C² node accumulates Round Complete messages and reports completed volleys to its battery C² node in "Volley Complete" messages which are defined as:

```
typedef struct
{
    UCHAR           return_code;        /* Mission complete            */
    USHORT          volley_count;       /* which volley e.g. 4 of  8   */
    USHORT          volley_sequence;    /* volleys in mission e.g. 8   */
    USHORT          number_of_active_guns;  /* num of guns firing      */
} VOLLEY_COMPLETE_MESSAGE;
```

The platoon C² node produces the SIMNET Indirect Fire PDU which reports up to five indirect fire detonations and their locations. The battery C² node also produces an "Indirect Fire Volley" PDU from each Volley Complete message. The Indirect Fire Volley PDU is sent through the Eagle CGF Manager to the SIU where it is available to the Eagle model. The Eagle model incorporates Indirect Fire Volleys in its damage assessment routines. The Indirect Fire Volley PDU is defined as:

```
typedef struct
{
    unsigned long    unit_id;           /* Firing unit's id            */
    OBJECT_TYPE      weapon_fired;      /* Weapon fired                */
    OBJECT_TYPE      munition;          /* Munition type being fired   */
    EAGLE_LOCATION   target_loc;        /* Fire at location            */
    unsigned long    width;             /* Sheath width                */
    unsigned long    depth;             /* Sheath depth                */
    unsigned short   orientation;       /* Sheath orientation (deg.)   */
    unsigned short   rounds;            /* Number of rounds fired      */
} INDIRECT_FIRE_VOLLEY;
```

4. Design and implement indirect fire production from CGF Testbed vehicles. **[In Progress]**
In conjunction with Task 1a - Add Vehicle Types, the M109 Self Propelled Artillery vehicle is being added to the IST CGF Testbed. Changes are required in both components of the CGF Testbed. First, the OI is being modified to create M109s and, as described above, the facility to give Battery Fire Missions orders is being added to the OI. Second, the M109 firing behavior is being added to the Simulator component. The M109 firing behavior is under the control of an FSM which is started with the receipt of a Gun Fire Mission message (see subtask 3 above). This FSM has the following six states:

START	Initialize data structures.
STOP_MOVEMENT	Stop movement.
FACE_TARGET_LOC	Emplace the gun facing towards the target location.
ELEVATE_GUN	Elevate the tube.
SHOOT	Produce SIMNET Fire PDU and Round Complete message.
DONE	If mission complete, quit otherwise go to SHOOT.

5. Design IOP Indirect Fire Volley PDU. **[In Progress]**
   See subtask 3 above for definition of Indirect Fire Volley PDU.

6. Design and implement algorithm to consolidate BDS-D Indirect Fire PDUs into IOP Indirect Fire Volley PDU. **[In Progress]**
   As described in subtask 3 above, the battery C^2 node creates IOP Indirect Fire Volley PDUs from the information in the Volley Complete messages received from platoon C^2 nodes.

7. Coordinate necessary changes to SIU with LANL. **[In Progress]**
   In progress.

8. Define test scenario with TRAC and LANL. **[Not Begun]**
   To be completed at next TIM in December 1993 or January 1994.

9. Test functionality with test scenario and correct errors as needed. **[Not Begun]**
   To be completed during or immediately after next TIM in December 1993 or January 1994.

2.11 Task 19: Allow partial disaggregation

description

Allow the partial disaggregation of the individual entities of an Eagle unit, such as helicopters, planes, or RPVs.

Partial disaggregation would allow portions of aggregated units to participate in disaggregated combat without necessitating the disaggregation of the entire unit. Thus disaggregation would no longer be an "all or nothing" operation on a particular unit.

subtasks

i. Modify Disaggregation Request PDU to allow single vehicle to be specified without unit. [**Completed**]
ii. Modify Disaggregation Process to disaggregate (i.e. create) a single vehicle. [**Completed**]
iii. Develop with TRAC a scenario that incorporates a single vehicle disaggregation. [**Not Begun**]
iv. Coordinate necessary modifications of SIU with LANL. [**Not Begun**]
v. Define test scenario with TRAC and LANL. [**Not Begun**]
vi. Test functionality with test scenario and correct errors as needed. [**Not Begun**]

progress on subtasks

i. Modify Disaggregation Request PDU to allow single vehicle to be specified without unit. [**Completed**]
The type "Any_Individual_Vehicle" is being added to the enumeration of unit types that may be disaggregated (see appendix E).

ii. Modify Disaggregation Process to disaggregate (i.e. create) a single vehicle. [**Completed**]
The composition grammar developed earlier in task #4 is sufficiently general to allow single vehicles to be specified within a unit designated for single vehicle disaggregations. A "Any_Individual_Vehicle" unit has been created and the vehicle types allowed in individual vehicle disaggregations have been made its members. The disaggregation process requires minimal revisions to accommodate individual vehicle disaggregations.

iii. Develop with TRAC a scenario that incorporates a single vehicle disaggregation. [**Not Begun**]
To be completed at next Technical Interchange Meeting in December 1993 or January 1994.

iv. Coordinate necessary modifications of SIU with LANL. [**Not Begun**]
In progress.

v. Define test scenario with TRAC and LANL. [**Not Begun**]
To be done at next Technical Interchange Meeting in December 1993 or January 1994.

vi. Test functionality with test scenario and correct errors as needed. [**Not Begun**]
To be performed after next Technical Interchange Meeting in December 1993 or January 1994.

3.0 Summary

Currently, four of the ECP1 tests are complete: tasks #2 Tracking Eagle Units in the CGF Testbed, #3 Base Eagle Unit Detection on Line of Sight, #4 Platoon and Battalion Disaggregation, and #5 Include Manned Simulators.

An extension to task #2 in combination with task #7 Make Eagle Units Visible on Stealth called task #2k Detailed Unit Appearance is being tested. Task 2k displays individual vehicles in a templated formation around Eagle units' centers of mass.

Task #10 Full Call for Fire is awaiting testing with Eagle and the SIU.

Two tasks, task #14 Indirect Fire at Eagle Units and task #19 Partial Disaggregation are nearing completion. Task #14 is serving as a training task for two new software engineers and is expected to be completed approximately two weeks after its scheduled completion date.

Task #8 Control Stealth from OI has been started and is on schedule.

Finally, task #1a Add Vehicle types has been started with the addition of the M109 howitzer needed by task #14.

This project is currently one month behind schedule due to task #2k (the extension to task #2) but is approximately $80,000 under budget. The project is now fully staffed with trained software engineers; in fact, for the remainder of the project there will be three instead of the budgeted two software engineers on the project. This staffing will allow IST to bring the project will be back on schedule within the next three months. There is adequate personnel funds in the project to support this extra effort.

Appendix A : Partial Listing of objects.h

This appendix shows a partial listing of the SIMNET definitions for echelons. These definitions are used in task (2) Tracking Eagle Units in the CGF Testbed. These definitions constrain the options available for echelon definition within SIMNET.

OBJECTS.H

```
/*** CONSTANTS ***/

/*  Number of bit positions for shift to determine posture.        */
#define LF_POSITION_SHIFT 27

/* The high-order three bits identify an object type's domain: */
#define OBJ_DOMAIN_MASK         0xE0000000
#define OBJ_DOMAIN_SHIFT        29
#define OBJ_DOMAIN_OTHER        (0L << OBJ_DOMAIN_SHIFT)
#define OBJ_DOMAIN_VEHICLE      (1L << OBJ_DOMAIN_SHIFT)
#define OBJ_DOMAIN_MUNITION     (2L << OBJ_DOMAIN_SHIFT)
#define OBJ_DOMAIN_STRUCTURE    (3L << OBJ_DOMAIN_SHIFT)
#define OBJ_DOMAIN_LIFEFORM     (4L << OBJ_DOMAIN_SHIFT)
#define OBJ_DOMAIN_ECHELON      (5L << OBJ_DOMAIN_SHIFT)

/* --- Echelon sceme --- */

/* The next three bits of a echelon type identify its environment: */
#define ECH_ENVIRONMENT_MASK       0x1C000000
#define ECH_ENVIRONMENT_SHIFT      26
#define ECH_ENVIRONMENT_AIR        (1L << ECH_ENVIRONMENT_SHIFT)
#define ECH_ENVIRONMENT_GROUND     (2L << ECH_ENVIRONMENT_SHIFT)
#define ECH_ENVIRONMENT_SPACE      (3L << ECH_ENVIRONMENT_SHIFT)
#define ECH_ENVIRONMENT_WATER      (4L << ECH_ENVIRONMENT_SHIFT)

/* The next three bits of an echelon type identify its class: */
#define ECH_CLASS_MASK          0x03800000
#define ECH_CLASS_SHIFT         23
#define ECH_CLASS_OTHER         (0L << ECH_CLASS_SHIFT)

/* Classes of air echelons: */
#define ECH_CLASS_FIXED_WING          (1L << ECH_CLASS_SHIFT)
#define ECH_CLASS_LIGHTER_THAN_AIR    (2L << ECH_CLASS_SHIFT)
#define ECH_CLASS_ROTARY_WING         (3L << ECH_CLASS_SHIFT)

/* Classes of ground echelons: */
#define ECH_CLASS_SP_ARMORED_TRACKED     (1L << ECH_CLASS_SHIFT)
#define ECH_CLASS_SP_ARMORED_WHEELED     (2L << ECH_CLASS_SHIFT)
#define ECH_CLASS_SP_UNARMORED_TRACKED   (3L << ECH_CLASS_SHIFT)
#define ECH_CLASS_SP_UNARMORED_WHEELED   (4L << ECH_CLASS_SHIFT)
#define ECH_CLASS_TOWED                  (5L << ECH_CLASS_SHIFT)

/* The next six bits of a echelon type identify its country of origin: */
#define ECH_COUNTRY_MASK        0x007E0000
#define ECH_COUNTRY_SHIFT       17
#define ECH_COUNTRY_OTHER       (COUNTRY_OTHER+0L << ECH_COUNTRY_SHIFT)
#define ECH_COUNTRY_US          (COUNTRY_US+0L << ECH_COUNTRY_SHIFT)
```

```
#define ECH_COUNTRY_USSR        (COUNTRY_USSR+0L << ECH_COUNTRY_SHIFT)
#define ECH_COUNTRY_GERMANY     (COUNTRY_GERMANY+0L << ECH_COUNTRY_SHIFT)

/* The next five bits of an echelon type identify the type of echelon: */
#define ECH_KIND_MASK           0x0001F800
#define ECH_KIND_SHIFT          11
#define ECH_KIND_OTHER          (0L << ECH_KIND_SHIFT)

/* The kinds of land echelons: */
#define ECH_KIND_SINGLE             (1L << ECH_KIND_SHIFT)
#define ECH_KIND_SQUAD              (2L << ECH_KIND_SHIFT)
#define ECH_KIND_SECTION            (3L << ECH_KIND_SHIFT)
#define ECH_KIND_PLATOON            (4L << ECH_KIND_SHIFT)
#define ECH_KIND_BATTERY            (5L << ECH_KIND_SHIFT)
#define ECH_KIND_COMPANY            (6L << ECH_KIND_SHIFT)
#define ECH_KIND_BATTALION          (7L << ECH_KIND_SHIFT)
#define ECH_KIND_REGIMENT           (8L << ECH_KIND_SHIFT)
#define ECH_KIND_BRIGADE            (9L << ECH_KIND_SHIFT)
#define ECH_KIND_DIVISION           (10L << ECH_KIND_SHIFT)
#define ECH_KIND_CORPS              (11L << ECH_KIND_SHIFT)
#define ECH_KIND_NEW_ARMY_CORPS     (12L << ECH_KIND_SHIFT)
#define ECH_KIND_ARMY               (13L << ECH_KIND_SHIFT)
#define ECH_KIND_ARMY_GROUP         (14L << ECH_KIND_SHIFT)
#define ECH_KIND_FRONT              (15L << ECH_KIND_SHIFT)
#define ECH_KIND_BATTALION_HQ       (16L << ECH_KIND_SHIFT)
#define ECH_KIND_REGIMENT_HQ        (17L << ECH_KIND_SHIFT)
#define ECH_KIND_BRIGADE_HQ         (18L << ECH_KIND_SHIFT)
#define ECH_KIND_DIVISION_HQ        (19L << ECH_KIND_SHIFT)
#define ECH_KIND_CORPS_HQ           (20L << ECH_KIND_SHIFT)
#define ECH_KIND_NEW_ARMY_CORPS_HQ  (21L << ECH_KIND_SHIFT)
#define ECH_KIND_ARMY_HQ            (22L << ECH_KIND_SHIFT)
#define ECH_KIND_ARMY_GROUP_HQ      (23L << ECH_KIND_SHIFT)
#define ECH_KIND_FRONT_HQ           (24L << ECH_KIND_SHIFT)

/* The kinds of air echelons: */
#define ECH_KIND_SORTIE             (1L << ECH_KIND_SHIFT)
#define ECH_KIND_FLIGHT_OF_TWO      (2L << ECH_KIND_SHIFT)
#define ECH_KIND_FLIGHT_OF_THREE    (3L << ECH_KIND_SHIFT)
#define ECH_KIND_FLIGHT_OF_FOUR     (4L << ECH_KIND_SHIFT)
#define ECH_KIND_AIR_PLATOON        (5L << ECH_KIND_SHIFT)
#define ECH_KIND_AIR_COMPANY        (6L << ECH_KIND_SHIFT)
#define ECH_KIND_AIR_CORPS_HQ       (7L << ECH_KIND_SHIFT)

/* The last five bits of a echelon type identify its function: */
#define ECH_FUNC_MASK           0x0000001F
#define ECH_FUNC_SHIFT          0
#define ECH_FUNC_MISCELLANEOUS  0       /* miscellaneous        */

/* Functions of air echelons: */
#define ECH_FUNC_AIR_COMBAT         1   /* air combat           */
#define ECH_FUNC_GROUND_ATTACK      2   /* ground attack        */
#define ECH_FUNC_AIR_RECON          3   /* reconnaissance       */
#define ECH_FUNC_AIR_INFANTRY       4
#define ECH_FUNC_AIR_ASSAULT_INFANTRY 5
```

```
/* Functions of ground echelons: */
#define ECH_FUNC_ANTI_AIRCRAFT        1      /* AA-gun or SAM        */
#define ECH_FUNC_PERSONNEL_CARRIER    2      /* armored PC           */
#define ECH_FUNC_COMMAND_POST         3      /* command post         */
#define ECH_FUNC_HOWITZER             4      /* howitzer             */
#define ECH_FUNC_MORTAR               5      /* mortar               */
#define ECH_FUNC_ROCKET_LAUNCHER      6      /* mult. launcher       */
#define ECH_FUNC_GROUND_RECON         7      /* reconnaissance       */
#define ECH_FUNC_RECOVERY             8      /* recovery             */
#define ECH_FUNC_SUPPLY_TRUCK         9      /* supply truck         */
#define ECH_FUNC_TANK_DESTROYER      10      /* tank destroyer       */
#define ECH_FUNC_LIGHT_TANK          11      /* tank, light          */
#define ECH_FUNC_MAIN_BATTLE_TANK    12      /* tank, main battle    */
#define ECH_FUNC_INFANTRY            13      /* infantry             */
#define ECH_FUNC_ARMORED_CAVALRY     14      /* armored cavalry      */
#define ECH_FUNC_COMBAT_ENGINEERING  15      /* combat eng           */

/* Functions of sea echelons */
#define ECH_FUNC_AMMUNITION                1
#define ECH_FUNC_AMPHIB_ASSAULT            2
#define ECH_FUNC_AMPHIB_CARGO              3
#define ECH_FUNC_AMPHIB_COMMAND            4
#define ECH_FUNC_AMPHIB_TRANSPORT          5
#define ECH_FUNC_ATTACK_SUB                6
#define ECH_FUNC_BALLISTIC_MISSILE_SUB     7
#define ECH_FUNC_BATTLESHIP                8
#define ECH_FUNC_CARRIER                   9
#define ECH_FUNC_COMBAT_STORES            10
#define ECH_FUNC_CRUISER                  11
#define ECH_FUNC_DESTROYER_TENDER         12
#define ECH_FUNC_DESTROYER                13
#define ECH_FUNC_DOCK_LANDING             14
#define ECH_FUNC_FAST_COMBAT_SUPPORT      15
#define ECH_FUNC_FLEET_OILER              16
#define ECH_FUNC_FRIGATE                  17
#define ECH_FUNC_HYDROFOIL                18
#define ECH_FUNC_MINE_CM                  19
#define ECH_FUNC_OCEAN_MINESWEEP          20
#define ECH_FUNC_REPAIR                   21
#define ECH_FUNC_REPLENISHMENT            22
#define ECH_FUNC_SALVAGE                  23
#define ECH_FUNC_SEA_MISC                 24
#define ECH_FUNC_SUB_TENDER               25
#define ECH_FUNC_TANK_LANDING             26

/* Echelon appearance modifiers pertaining to specific types of echelon: */
#define ECH_IS_SUBSUMED       0x80000000
#define ECH_SIZE_MASK         0x18000000
#define ECH_SIZE_STANDARD     0x00000000
#define ECH_SIZE_HEAVY        0x08000000
#define ECH_SIZE_UNIT_LESS    0x10000000
```

*** End of file ***

Appendix B : Composition and Formations Grammars

Unit Composition Grammar:

```
<statement>      ::= <map-definition> | <unit-definition>

<map-definition> ::= 'Map' <map-name> := <map-expression> ;

<map-expression> ::= {
                       <template-name1> => <formation-var-name1>,
                       <template-name2> => <formation-var-name2>,
                       ...
                       <template-namek> => <formation-var-namek>
                     }

<unit-definition> ::= 'Unit' <unit-name> := { <unit-details>, <unit-details> ... } ;

<unit-details>   ::= 'Type'          := <unit-type> ;          |
                     'SubUnits'      := <unit-name1>, ... ;    |
                     'SubUnitMap'    := <map-name> ;           |
                     'Vehicles'      := <vehicle1> [ n ], ... ; |
                     'VehicleMap'    := <map-name> ;
```

Unit Formation Grammar:

```
<statement> ::= <assignment> | <print>

<assignment> ::= <var> := <expression>;

<print>     ::= print <var>, <var>, ..., <var>;

<expression> ::= <expression> <operator> | <point-set> | <var>

<var>       ::= <var_char> <var> | <var_char>

<point-set> ::= { <point> <point> ... <point> }

<point>     ::= ( <double> <double> )

<operator>  ::= &shift ( <double>, <double> ) |
                &rotate( <double> )          |
                &cat   ( <point-set> )       |
                &cat   ( <point> )           |
                &cat   ( <var> )             |
                &flipv                       |
                &fliph

<double>    ::= [-/+] digits.digits

<var_char>  ::= a | b | c | d | e | ... | z | _
```

Appendix C : Composition Definition

```
*************************************************************
*  Date Created: 07/12/93                                   *
*  The Eagle composition file.                              *
*************************************************************

*---------------------*
*** MAP DEFINITIONS ***
*---------------------*

MAP HQSecVehMap :=
{
    Assembly => HQSecVehAssembly,
    Wedge    => HQSecVehWedge,
    Vee      => HQSecVehVee,
    Line     => HQSecVehLine,
    Column   => HQSecVehColumn
};

MAP Plt_1VehMap :=
{
    Assembly => Plt_VehAssembly1,
    Wedge    => Plt_VehWedge1,
    Vee      => Plt_VehVee1,
    Line     => Plt_VehLine1,
    Column   => Plt_VehColumn1,
};

MAP Plt_2VehMap :=
{
    Assembly => Plt_VehAssembly1,
    Wedge    => Plt_VehWedge2,
    Vee      => Plt_VehVee2,
    Line     => Plt_VehLine1,
    Column   => Plt_VehColumn1,
};

MAP Artillery_Plt_VehMap :=
{
    Assembly => Plt_VehAssembly1,
    Wedge    => Plt_VehWedge1,
    Vee      => Plt_VehVee1,
    Line     => Plt_VehLine1,
    Column   => Plt_VehColumn1,
};

MAP IndividualVehicleMap :=
{
    Assembly => IndividualVehiclePoints,
    Wedge    => IndividualVehiclePoints,
    Vee      => IndividualVehiclePoints,
    Line     => IndividualVehiclePoints,
    Column   => IndividualVehiclePoints,
```

```
};

MAP CavTroopSubUnitMap :=
{
    Assembly => CavTroopSubUnitAssembly,
    Wedge    => CavTroopSubUnitWedge,
    Vee      => CavTroopSubUnitVee,
    Line     => CavTroopSubUnitLine,
    Column   => CavTroopSubUnitColumn
};

MAP BattalionSubUnitMap :=
{
    Assembly => BattalionSubUnitAssembly,
    Wedge    => BattalionSubUnitWedge,
    Vee      => BattalionSubUnitVee,
    Line     => BattalionSubUnitLine,
    Column   => BattalionSubUnitColumn
};

MAP BlueBatterySubUnitMap :=
{
    Assembly => BatterySubUnitAssembly,
    Wedge    => BatterySubUnitWedge,
    Vee      => BatterySubUnitVee,
    Line     => BatterySubUnitLine,
    Column   => BatterySubUnitColumn
};

*-------------------------------------*
*** UNIT DEFINITIONS ***
*-------------------------------------*

UNIT Blue_Plt_1 :=
{
    Level      := Platoon;
    SubUnits   := ;
    SubUnitMap := ;
    Vehicles   := M2 [ 4 ], DI_USA [ 4 ];
    VehicleMap := Plt_1VehMap;
}

UNIT Blue_Plt_2 :=
{
    Level      := Platoon;
    SubUnits   := ;
    SubUnitMap := ;
    Vehicles   := ABRAMS [ 4 ];
    VehicleMap := Plt_2VehMap;
}
```

```
UNIT Blue_Artillery_Plt :=
{
    Level      := Platoon;
    SubUnits   := ;
    SubUnitMap := ;
    Vehicles   := M109 [ 4 ];
    VehicleMap := Artillery_Plt_VehMap;
}

UNIT Red_Plt_1 :=
{
    Level      := Platoon;
    SubUnits   := ;
    SubUnitMap := ;
    Vehicles   := BMP [ 4 ], DI_USSR [ 4 ];
    VehicleMap := Plt_1VehMap;
}

UNIT Red_Plt_2 :=
{
    Level      := Platoon;
    SubUnits   := ;
    SubUnitMap := ;
    Vehicles   := T72 [ 4 ];
    VehicleMap := Plt_2VehMap;
}

UNIT Any_Individual_Vehicle :=
{
    Level      := Platoon;
    SubUnits   := ;
    SubUnitMap := ;
    Vehicles   := ABRAMS, M2, DI_USA, M109, T72, BMP, DI_USSR;
    VehicleMap := IndividualVehicleMap;
}

UNIT Blue_Cav_Troop :=
{
    Level      := Company;
    SubUnits   := Blue_Plt_1, Blue_Plt_1, Blue_Plt_2;
    SubUnitMap := CavTroopSubUnitMap;
    Vehicles   := ABRAMS, M2;
    VehicleMap := HQSecVehMap;
}

UNIT BLUE_BN_TF_1 :=
{
    Level      := Battalion;
    SubUnits   := Blue_Cav_Troop, Blue_Cav_Troop;
    SubUnitMap := BattalionSubUnitMap;
    Vehicles   := ;
    VehicleMap := ;
}
```

```
UNIT Blue_Battery_1 :=
{
    Level      := Battery;
    SubUnits   := Blue_Artillery_Plt, Blue_Artillery_Plt;
    SubUnitMap := BlueBatterySubUnitMap;
    Vehicles   := ;
    VehicleMap := ;
}

UNIT Red_Cav_Troop :=
{
    Level      := Company;
    SubUnits   := Red_Plt_1, Red_PLT_1, Red_Plt_2;
    SubUnitMap := CavTroopSubUnitMap;
    Vehicles   := T72, BMP;
    VehicleMap := HQSecVehMap;
}

UNIT Red_BN_TF_1 :=
{
    Level      := Battalion;
    SubUnits   := Red_Cav_Troop, Red_Cav_Troop;
    SubUnitMap := BattalionSubUnitMap;
    Vehicles   := ;
    VehicleMap := ;
}

*** END OF FILE ***
```

Appendix D : Formation Definition

```
*****************************************************************************
*  Date Created: 07/12/93                                                   *
*  The Eagle formation file.                                                *
*****************************************************************************

CavTroopSubUnitAssembly := { (0 75) (-200 0) (200 -75) };
CavTroopSubUnitWedge    := { (0 75) (-200 -75) (200 -75) };
CavTroopSubUnitVee      := CavTroopSubUnitWedge &flipv;
CavTroopSubUnitLine     := { (0 0) (-400 0) (400 0) };
CavTroopSubUnitColumn   := { (0 220) (0 0) (0 -220) };

BattalionSubUnitAssembly := { ( 250 250 ) ( -250 -250 ) };
BattalionSubUnitWedge    := { ( 250 250 ) ( -250 -250 ) };
BattalionSubUnitVee      := BattalionSubUnitWedge &flipv;
BattalionSubUnitLine     := { ( 250 250 ) ( -250 -250 ) };
BattalionSubUnitColumn   := { ( 250 250 ) ( -250 -250 ) };

BatterySubUnitAssembly := { ( -100 0 ) ( 100 -75 ) };
BatterySubUnitWedge    := { ( -200 -75 ) ( 200 -75 ) };
BatterySubUnitVee      := BatterySubUnitWedge &flipv;
BatterySubUnitLine     := { ( -200 0 ) ( 200 0 ) };
BatterySubUnitColumn   := { ( 0 120 ) ( 0 -120 ) };

*-----------------------*
*** Platoon Assembly: ***
*-----------------------*
HQSecVehAssembly := { (0 0) (0 100) };
Plt_VehAssembly1 := { (-50 0) (-150 0) (50 0) (150 0)
                      (-40 0) (-140 0) (60 0) (160 0) };

*-----------------------*
*** Platoon wedge:    ***
*-----------------------*
HQSecVehWedge := { (0 0) (0 -100) };
Plt_VehWedge1 := { (0 50) (50 0) (-50 0) (-100 -50)
                   (10 50) (60 0) (-40 0) (-90 -50) };
Plt_VehWedge2 := Plt_VehWedge1 &fliph;

*-----------------------*
*** Platoon vee:      ***
*-----------------------*
HQSecVehVee := HQSecVehWedge &flipv;
Plt_VehVee1 := Plt_VehWedge1 &flipv;
Plt_VehVee2 := Plt_VehWedge2 &flipv;

*-----------------------*
*** Platoon line:     ***
*-----------------------*
HQSecVehLine := { (0 -50) (100 -50) };
Plt_VehLine1 := { (-50 0) (-150 0) (50 0) (150 0)
                  (-40 0) (-140 0) (60 0) (160 0) };
```

```
*-----------------------*
*** Platoon column:   ***
*-----------------------*
HQSecVehColumn := { (0 110) (0 -110) };
Plt_VehColumn1 := { (0 50) (0 25) (0 -25) (0 -50)
                    (10 50) (10 25) (10 -25) (10 -50) };

*--------------------------------------*
*** Individual Vehicle All Formation ***
*--------------------------------------*
IndividualVehiclePoints := { ( 0 0 ) ( 0 0 ) ( 0 0 ) ( 0 0 ) ( 0 0 ) ( 0 0 ) };

*------------------------*
*** Used for Debugging ***
*------------------------*
*Print Plt_VehColumn1, Plt_VehLine1;

*** End of file ***
```

Appendix E : Partial Listing of loc_epro.h

This file contains the definitions of all the IOP messages. It is included in this report for completeness.

```
#ifndef __LOC_EPRO
#define __LOC_EPRO

/*****************************************************************************
*  Header Scope:    Local to module Eagle                                     *
*                                                                             *
*  The Eagle module prototypes and declarations needed in the Eagle and       *
*  Protocol modules.                                                          *
*                                                                             *
*  Last Changed:  6.412                                                       *
*  Responsibility: Clark R. Karr                                              *
*****************************************************************************/

/*** INCLUDES ***/

#include "eagle.h"
#include "exec.h"
#include "message.h"

/*** CONSTANTS ***/

#define SIU_VEHICLE_NUMBER    0
#define MAX_UNIT_NAME_LENGTH  20

/*  The Eagle trace output level settings    */
enum
{
    NO_MESSAGES,
    ERROR,
    DEMO,
    DETAIL_DEMO,
    DEBUG,
    ALL_MESSAGES
};


/* The types of alignments used when dealing with Disaggregation Templates */
/* ( ENUM_EGL_ALIGNMENT_TYPES )                                            */
enum
{
    BLUE_FORCE,
    RED_FORCE,
    MAX_ALIGNMENT,
    ALL_FORCES = MAX_ALIGNMENT
};
```

```
/* Echelon Levels                */
/* ( ENUM_EGL_ECHELON_LEVELS )   */
enum
{
    CORPS,
    DIVISION,
    BRIGADE,
    REGIMENT,
    BATTALION,
    BATTERY,
    COMPANY,
    PLATOON,
    ELEMENT,
    INDIVIDUAL_VEHICLE,
    ECHELON_UNKNOWN,
    MAX_ECHELON_LEVEL,
};

/* Echelon types that can be disaggregated. */
enum
{
    BN_TF_1,
    CAVALRY_TROOP,
    PLATOON_1,
    BATTERY_1,
    SINGLE_VEHICLE,
    MAX_DISAGG_ECHELON_TYPES
};

/* The types of echelons used when dealing */
/* with the unit appearance pdus.          */
/* ( ENUM_EGL_ECHELON_TYPES )              */
enum
{
    CAV_TROOP,
    ARMOR,
    INFANTRY,
    MECH_INFANTRY,                  /* This is "MECH-INF" in eagle. */
    CAVALRY,
    ARMORED_CAVALRY,
    ARTILLERY,
    SELF_PROPELLED_ARTILLERY,
    CAS,
    ENGINEER,
    ADA,
    ANTI_TANK,
    ARMY_AVIATION_FW,
    ARMY_AVIATION_RW,
    ARMY_ATTACK_HELICOPTER,
    AIR_CAVALRY,
    MOTORIZED_RIFLE,
    ARMOR_HEAVY_TASK_FORCE,
    MECHANIZED_HEAVY_TASK_FORCE,
```

```
    COMMAND_POST,
    CEWI,
    TANK_ONLY,                          /* This is a "TANK" in eagle. */
    MAX_ECHELON_TYPE
};

/* The types of templates used when dealing with Disaggregation Templates  */
/* ( ENUM_EGL_TEMPLATE_TYPES )                                              */
enum
{
    ASSEMBLY,
    VEE,
    WEDGE,
    LINE,
    COLUMN,
    MAX_TEMPLATE
};

/* PDU types within Eagle Protocol packets:                                 */
/* ( ENUM_EGL_PDU_SUB_TYPES )                                               */
enum
{
    INITIALIZE_VARIANT,
    START_VARIANT,
    STOP_VARIANT,
    RESUME_VARIANT,
    PAUSE_VARIANT,
    STATUS_REQ_VARIANT,

    DISAGG_REQ_VARIANT,
    DISAGG_RESP_VARIANT,
    AGG_REQ_VARIANT,

    OP_ORDER_VARIANT,
    ADV_OP_ORDER_VARIANT,
    OPERATOR_INTENT_VARIANT,

    UNIT_APPEARANCE_VARIANT,
    DETAILED_APPEARANCE_VARIANT,

    CALL_FOR_FIRE_VARIANT,

    BATTERY_FIRE_MISSION_VARIANT,
    PLATOON_FIRE_MISSION_VARIANT,
    GUN_FIRE_MISSION_VARIANT,

    ROUND_COMPLETE_VARIANT,
    VOLLEY_COMPLETE_VARIANT,

    INDIRECT_FIRE_VOLLEY_VARIANT,

    INDIRECT_FIRE_REQ_VARIANT,
    TEMP_INDIRECT_FIRE_VARIANT,
```

```
    EPRO_GENERIC_MESSAGE_END
};

/* Eagle Message types corresponding to a subset                    */
/* of the above Protocol variants:                                  */
/* ( ENUM_EGL_CGF_INTERNAL_MSG_TYPES )                              */
enum
{
    DISAGGREGATE = GENERIC_MESSAGE_END,
    AGGREGATE,
    OPERATION_ORDER,
    ADV_OPERATION_ORDER,
    OPERATOR_INTENT_REQUEST,
    INDIRECT_FIRE_REQUEST,
    BATTERY_FIRE_MISSION_REQUEST,
    GUN_FIRE_MISSION_REQUEST,
    ROUND_COMPLETE_REQUEST,
    VOLLEY_COMPLETE_REQUEST,
    PLATOON_FIRE_MISSION_REQUEST,
    INDIRECT_FIRE_VOLLEY_REQUEST,
    TEMP_INDIRECT_FIRE_REQUEST,
    IDENTIFY_SIU,
    UNIT_APPEARANCE,
    DETAILED_UNIT_APPEARANCE,
    EAGLE_GENERIC_MESSAGE_END
};

/* Types of nodes sending and receiving Eagle Protocol PDUs:        */
/* ( ENUM_EGL_NODE_TYPES )                                          */
enum
{
    SIU,                /* Simulation Integration Unit */
    EAGLE_CGF_MANAGER   /* Eagle Manager               */
};

/* Operator Intents available:                                      */
/* ( ENUM_EGL_OP_INTENT_TYPES )                                     */
enum
{
    REQUEST_AGGREGATION,
    CHANGING_PHASE,
    CHANGING_TASK,
    PROCEEDING_NEXT_OBJ_NUM,
    PROCEEDING_NEXT_OBJ_LOC,
    CHANGING_OA,
    NUM_OF_OP_INTENTS
};
```

```
/* Types of Operational Activities for use in Operator Intent messages.     */
/* ( ENUM_EGL_OP_ACT_TYPES )                                                */
enum
{
    ROAD_MARCH,
    OCCUPY_HASTY_BATTLE_POSITION,
    MOVEMENT_TO_CONTACT,
    ASSAULTING_ENEMY,
    DELAY,
    DEFILE_CROSSING,
    OCCUPY_ASSEMBLY_AREA,
    OCCUPY_BATTLE_POSITION,
    APPROACHING_ENEMY,
    BREAK_CONTACT,
    INSTRIDE_BREACH,
    B_TAC_ROAD_MARCH_CPY,
    B_OCCUPY_HASTY_BATTLE_PSN_CPY,
    B_TRAVELING_OVERWATCH_CPY,
    B_DEFEAT_CPY,
    B_DELAY_CPY,
    B_DEFILE_CROSSING_CPY,
    B_OCCUPY_ASSEMBLY_AREA_CPY,
    B_OCCUPY_BATTLE_PSN_CPY,
    B_BOUNDING_OVERWATCH_CPY,
    B_BREAK_CONTACT_CPY,
    B_INSTRIDE_BREACH_CPY,
    B_DEFEND_ASSEMBLY_AREA_CPY,
    B_DEFEND_BATTLE_PSN_CPY,
    B_DEFEND_HASTY_BATTLE_PSN_CPY,
    B_DEFAULT,
    R_TAC_ROAD_MARCH_CPY,
    R_MARCH_CPY,
    R_OCCUPY_HASTY_BATTLE_PSN_CPY,
    R_PREBATTLE_FORMATION_CPY,
    R_ATTACK_FORMATION_CPY,
    R_DELAY_CPY,
    R_DEFILE_CROSSING_CPY,
    R_OCCUPY_CONCENTRATION_AREA_CPY,
    R_OCCUPY_BATTLE_PSN_CPY,
    R_BREAK_CONTACT_CPY,
    R_INSTRIDE_BREACH_CPY,
    R_DEFEND_CONCENTRATION_AREA_CPY,
    R_DEFEND_BATTLE_PSN_CPY,
    R_DEFEND_HASTY_BATTLE_PSN_CPY,
    R_DEFAULT,
    DEFAULT,
    NUM_OF_OP_ACTIVITIES
};
```

```
/* Effectiveness */
enum
{
    EFFECTIVE,
    MARG_EFFECTIVE,
    INEFFECTIVE
};

/* Point or sheath battery fire mission */
enum
{
    POINT_MISSION,
    SHEATH_MISSION
};

/*** TYPES ***/

/* For brevity. */
typedef unsigned char  UCHAR;
typedef unsigned long  ULONG;
typedef unsigned short USHORT;

/* Type of the ID by which a unit (company) is known. */
typedef unsigned long UNIT_ID;

/* The type of Eagle locations is different from Simulator locations. */
typedef struct
{
    unsigned long x, y, z;    /* A point on Eagle terrain */
} EAGLE_LOCATION;

/*  The Initialize PDU variant.  */
typedef struct
{
    ULONG           terrain_id;   /* Terrain database identifier */
    ULONG           start_time;   /* Start time of exercise      */
    VEHICLE_ID      sender_id;    /* Sender's id                 */
    UCHAR           exercise_id;  /* Exercise identifier         */
} INIT_EXERCISE;

/*  Defines START, STOP, RESUME, and PAUSE PDU variants. */
typedef struct
{
    UCHAR  exercise_id;           /* Exercise identifier */
} SIMULATION_CONTROL;
```

```
/* Data in Disaggregation Requests and Partial Disaggregation Requests. */
typedef struct
{
    UNIT_ID         unit_id;          /* Unit being disaggregated          */
    EAGLE_LOCATION  unit_loc;         /* Location of the unit's center     */
    USHORT          unit_heading;     /* Unit's direction of movement      */
    UCHAR           unit_alignment;   /* Side of unit                      */
                                      /*  (ENUM_EGL_ALIGNMENT_TYPES)       */
    UCHAR           echelon_level;    /* Echelon Level                     */
                                      /*  (ENUM_EGL_ECHELON_LEVELS)        */
    UCHAR           echelon_type;     /* Echelon Type                      */
                                      /*  (ENUM_EGL_ECHELON_TYPES)         */
    UCHAR           op_act;           /* Index into template lookup table. */
                                      /*  (ENUM_EGL_OP_ACT_TYPES)          */
    char            padding[2];       /* Ends structure on 4 byte boundary. */
} DISAGG_INFO;

/* The Disaggregation Request record. */
typedef struct
{
    DISAGG_INFO  unit_info;         /* Specifications for disaggregation    */
    UCHAR        system_count;      /* Number of groups of entities in unit */
    char         padding[3];        /* Forces data to 4 byte boundary       */
    char         data;              /* Array of 'systemCount' elements of   */
                                    /* System Info.                         */
} DISAGG_REQ;

/* The data part of the Disaggr. request contains an array of elements */
/* of the following type:                                              */
typedef struct
{
    ULONG          entity_type;      /* DI, ABRAMS, BMP, etc.              */
    int            num_of_entities;  /* Number of this type in the unit    */
    char           padding[2];       /* Keeps structure on 4 byte boundary. */
} SYSTEM_INFO;

/* The Unit Disaggregation Request record. */
typedef struct
{
    DISAGG_INFO  unit_info;           /* Specifications for disaggregation    */
    VEHICLE_ID   oi_sup_mgr_id;       /* Id of the OI Sup Manager             */
    VEHICLE_ID   man_sim_mgr_id;      /* Id of the Man Sim Manager            */
    UCHAR        system_count;        /* Number of groups of entities in unit */
    char         name_string[ MAX_UNIT_NAME_LENGTH ];
                                      /* Name used for naming vehicles.       */
    char         name_index;          /* Current length of the string.        */
    char         data;                /* Array of 'systemCount' elements of   */
                                      /* System Info.                         */
} UNIT_DISAGG_REQ;
```

```
/*  The response to a Disaggregation request.   */
/*  Contains vehicleIDs as data if successful.  */
typedef struct
{
    UNIT_ID unit_id;        /* unit being disaggregated                */
    UCHAR   success;        /* FALSE or TRUE                           */
    UCHAR   system_count;   /* Number of groups of entities in unit    */
    UCHAR   seq_length;     /* Number of PDU's in this Disagg Response */
    UCHAR   seq_number;     /* Sequence number of this PDU             */
    char    data;           /* array[ systemCount ] of                 */
                            /*  DISAGG_RESP_SYSTEM_INFO                */
} DISAGG_RESP;

/* The data part of the Disaggregation response contains an array of       */
/* elements of the following type:                                         */
typedef struct
{
    ULONG         entity_type;      /* DI, ABRAMS, BMP, etc.                */
    UCHAR         num_of_entities;  /* Number of this type in the unit.     */
    char          padding[3];       /* Forces data to 4 byte boundary       */
    char          data;             /* array[ num_of_entities ] of Vehicle  */
                                    /* ID                                   */
} DISAGG_RESP_SYSTEM_INFO;

typedef struct
{
    VEHICLE_ID veh_id;             /* The vehicle id                        */
    char       padding[2];         /* Forces next veh_id to 4 byte boundary */
} PADDED_VEHICLE_ID;

/* The Aggregation Request record: */
typedef struct
{
    UNIT_ID unit_id;                /* The unit to aggregate. */
} AGG_REQ;

/* Operation Order */
typedef struct
{
    UNIT_ID to_unit_id;
    char    seq_length;     /* number of PDUs in this Op Order          */
    char    seq_number;     /* sequence number of this PDU              */
    int     size_of_data;   /* length of text in this PDU               */
    char    order;          /* array[ size_of_data ] of char            */
} OP_ORDER;

/* Advance Operation Order - exactly similar to OP_ORDER: */
typedef OP_ORDER   ADV_OP_ORDER;
```

```
/*  Tactical activities for use in Operator Intent messsages  */
typedef struct
{
    int             activity;   /* Type of Activity (ENUM_EGL_OP_ACT_TYPES) */
    EAGLE_LOCATION  location;   /* Location of activity                     */
} OPERATIONAL_ACTIVITY;

/*  Tactical activities for use in Operator Intent PDUs  */
typedef struct
{
    int             activity;   /* Type of Activity (ENUM_EGL_OP_ACT_TYPES) */
    char            padding[2]; /* Force location onto 4 byte boundary      */
    EAGLE_LOCATION  location;   /* Location of activity                     */
} OPERATIONAL_ACTIVITY_PDU;

/* Operator Intent */
typedef struct
{
    UNIT_ID     from_unit_id;     /* Originating unit                        */
    int         type;             /* Type of ( ENUM_EGL_OP_INTENT_TYPES )    */
    union
    {
        int                   phase;              /* Operation Phase num    */
        int                   task;               /* Operation Task num     */
        int                   objective;          /* Operation Objective    */
        EAGLE_LOCATION        location;           /* Operation location     */
        OPERATIONAL_ACTIVITY  change_activity_to; /* One of:                */
                                                  /*  ENUM_EGL_OP_ACT_TYPES*/
    } variant;
} OPERATOR_INTENT;

/* Operator Intent PDU */
typedef struct
{
    UNIT_ID  from_unit_id;     /* Originating unit                           */
    int      type;             /* Type of ( ENUM_EGL_OP_INTENT_TYPES )       */
    char     padding[2];       /* Force union onto 4 byte boundary           */
    union
    {
        int                       phase;              /* Operation Phase num  */
        int                       task;               /* Operation Task num   */
        int                       objective;          /* Operation Objective  */
        EAGLE_LOCATION            location;           /* Operation location   */
        OPERATIONAL_ACTIVITY_PDU  change_activity_to; /* One of:              */
                                                      /* ENUM_EGL_OP_ACT_TYPES */
    } variant;
} OPERATOR_INTENT_PDU;
```

```
/* Internal Indirect Fire Request */
typedef struct
{
    UNIT_ID         requesting_unit;  /* Unit making request              */
    UNIT_ID         target_unit;      /* Unit being attacked              */
    EAGLE_LOCATION  target_loc;       /* Fire at location                 */
} INDIRECT_FIRE_REQ;

/* Indirect Fire Request PDU */
typedef struct
{
    UNIT_ID         requesting_unit;  /* Unit making request              */
    UNIT_ID         target_unit;      /* Unit being attacked              */
    EAGLE_LOCATION  target_loc;       /* Fire at location                 */
} INDIRECT_FIRE_REQ_PDU;

/* Temporary Indirect Fire PDU                                            */
typedef struct
{
    ULONG           firing_system;  /* Type of weapon shooting            */
    OBJECT_TYPE     munition;       /* Munition type being fired          */
    UCHAR           quantity;       /* 1 through maximum ifire detonations*/
    char            padding[3];     /* Force location to 4 byte boundary  */
    EAGLE_LOCATION location[1];     /* actually location[1..quantity]     */
} TEMP_INDIRECT_FIRE;

/* Unit Appearance PDU                                                    */
typedef struct
{
    ULONG           vehicle;            /* Vehicle Id to be used in VAPDU.*/
    EAGLE_LOCATION  loc;                /* Location of the unit.          */
    float           heading;            /* The units heading (degrees).   */
    float           orientation;        /* The unit facing direction.     */
                                        /*     (degrees).                 */
    float           speed;              /* The unit speed (m/s)           */
    UCHAR           echelon_level;      /* Level of echelon               */
                                        /*  (e.g. Company, Btln, etc.)    */
    UCHAR           echelon_type;       /* Echelon type                   */
                                        /*  (e.g. Infantry, Armored, etc.)*/
    UCHAR           alignment;          /* Side of unit                   */
                                        /*  (ENUM_EGL_ALIGNMENT_TYPES)    */
    UCHAR           effectiveness;      /* Current effectiveness          */
    USHORT          relative_strength;  /* Precent active vehicles        */
    USHORT          operation_activity; /* Opertaional Activity           */
    UCHAR           hq_or_cp;           /* Headquarters or command post.  */
    UCHAR           remove;             /* Remove unit from exercise      */
    char            unit_name[MAX_UNIT_NAME_LENGTH];
} UNIT_APPEARANCE_PDU;
```

```
typedef struct
{
    ULONG           start_vehicle_num; /* Vehicle num to be used in VAPDU's */
    EAGLE_LOCATION  loc;               /* Location of unit.                 */
    float           heading;           /* The units heading (degrees)       */
    float           orientation;       /* The units orientation (degrees)   */
    float           speed;             /* The unit speed (m/s)              */
    UCHAR           echelon_level;     /* Level of echelon                  */
    UCHAR           echelon_type;      /* Echelon type                      */
    UCHAR           alignment;         /* Side of unit                      */
    UCHAR           effectiveness;     /* Current effectiveness             */
    USHORT          operation_activity;/* Operational Activity              */
    USHORT          remove;            /* Remove unit from exercise         */
    char            unit_name[MAX_UNIT_NAME_LENGTH];
    char            system_count;      /* Number of groups of entities in   */
                                       /* the unit                          */
    char            padding[2];        /* Force alignment to 4 byte boundry */
    char            data;              /* Array of 'sysCount' elements of   */
                                       /* System Info.                      */
} DETAILED_UNIT_INFO;

/* Detailed Unit Appearance PDU. */
typedef struct
{
    DETAILED_UNIT_INFO   unit_info;    /* Specifications for disaggregation */
    UCHAR                system_count; /* Number of groups of entities in   */
                                       /* unit.                             */
    char                 padding[3];   /* Forces data to 4 byte boundry.    */
    char                 data;         /* Array of 'systemCount' elements   */
                                       /* of System Info.                   */
} DETAILED_UNIT_APPEARANCE_PDU;

/* Indirect Fire Volley PDU                                                 */
typedef struct
{
    EAGLE_LOCATION  target_loc;     /* Fire at location.                    */
    ULONG           width;          /* Sheath width                         */
    ULONG           depth;          /* Sheath depth                         */
    USHORT          orientation;    /* Sheath orientation (degrees)         */
    USHORT          rounds;         /* Number of rounds fired               */
    OBJECT_TYPE     munition;       /* Munition type being fired            */
} INDIRECT_FIRE_VOLLEY;
```

```
/* Call for Fire PDU                                                    */
typedef struct
{
    EAGLE_LOCATION    taret_loc;        /* Fire at location              */
    UCHAR             effect;           /* Desired effect of fire        */
    UCHAR             target_ech_level; /* Target echelon level          */
                                        /*  (e.g. Company, Btln, etc.)   */
    UCHAR             target_ech_type;  /* Target echelon type           */
                                        /*  (e.g. Armor, Infantry, etc.) */
    UCHAR             cover;            /* Terrain cover                 */
    UCHAR             vegetation;       /* Vegetation around target      */
} CALL_FOR_FIRE;

typedef struct
{
    UNIT_ID           unit_id;          /* Requesting unit_id            */
    OBJECT_TYPE       munition;          /* Type of munition              */
    EAGLE_LOCATION    target_loc;        /* Target loc for Battery Fire   */
    USHORT            number_of_volleys; /* No of volleys to be fired     */
    UCHAR             point_or_sheath;   /* Point or sheath fire          */
    USHORT            width;             /* Width of the sheath           */
    USHORT            depth;             /* Depth of the sheath           */
    USHORT            orientation;       /* Orientation of the sheath     */
} BATTERY_FIRE_MISSION;

typedef struct
{
    float  x_vector;
    float  y_vector;
} VOLLEY_VECTOR;

typedef struct
{
    float x;
    float y;
} GUN_LOCATION;

typedef struct
{
    USHORT         mission_id;
    OBJECT_TYPE    munition;
    USHORT         number_of_volleys;
    VOLLEY_VECTOR  volley_advance_vector;
    USHORT         number_of_gun_locations;
    char           data;
} PLATOON_FIRE_MISSION;
```

```
typedef struct
{
    UCHAR    return_code;
    USHORT   volley_count;
    USHORT   volley_sequence;
    USHORT   number_of_active_guns;
} VOLLEY_COMPLETE_MESSAGE;

typedef struct
{
    VEHICLE_ID        gun_id;
    OBJECT_TYPE       munition;
    USHORT            number_of_volleys;
    GUN_LOCATION      gun_target_initial_loc;
    VOLLEY_VECTOR     volley_advance_vector;
} GUN_FIRE_MISSION;

typedef struct
{
    VEHICLE_ID        gun_id;
    GUN_LOCATION      detonation_loc;
    USHORT            round_complete;
    USHORT            round_sequence;
    UCHAR             gun_status;
}ROUND_COMPLETE_MESSAGE;

/* Handshaking messages:                                              */
typedef struct
{
    VEHICLE_ID source_id;           /* Caller identifier              */
    UCHAR     source_host_type;     /* Type of caller:                */
                                    /*  ( ENUM_EGL_NODE_TYPES )       */
    UCHAR     respondent_host_type; /* To whom? (ENUM_EGL_NODE_TYPES ) */
} STATUS_REQ;

/* Header of an Eagle Protocol packet: */
typedef struct
{
    VEHICLE_ID source;            /* The sender's ID                  */
    VEHICLE_ID destination;       /* The receiver's ID                */
    UCHAR      exercise;          /* The exercise ID                  */
    char       response_required; /* Is a response required?          */
    int        type;              /* PDU subtype (ENUM_EGL_PDU_SUB_TYPES) */
    int        data_length;       /* Number of bytes of data          */
    char       padding[2];        /* Terminates header on 4 byte boundary */
} EAGLE_HEADER;
```

```
/* Eagle Protocol Data Unit: */
typedef struct
{
    EAGLE_HEADER hdr;
    union
    {
        INIT_EXERCISE          initialize;          /* Init exercise PDU.    */
        SIMULATION_CONTROL     start;               /* Start exercise PDU.   */
        SIMULATION_CONTROL     stop;                /* Stop exercise PDU.    */
        SIMULATION_CONTROL     resume;              /* Resume exercise PDU.  */
        SIMULATION_CONTROL     pause;               /* Pause exercise PDU.   */

        STATUS_REQ             status_req;          /* Handshaking message   */

        DISAGG_REQ             disagg_req;          /* Disagg request        */
        DISAGG_RESP            disagg_resp;         /* Disagg response       */

        AGG_REQ                agg_req;             /* Aggregation request   */

        OP_ORDER               op_order;            /* Operation order       */
        ADV_OP_ORDER           adv_op_order;        /* Advance operation     */
                                                    /* order                 */
        OPERATOR_INTENT_PDU    op_intent;           /* Operator intent       */

        UNIT_APPEARANCE_PDU    unit_appearance;     /* Unit appearance       */

                                                    /* Detailed unit appr.   */
        DETAILED_UNIT_APPEARANCE_PDU detailed_unit_appearance;

        CALL_FOR_FIRE          call_for_fire;       /* Call for fire         */
        INDIRECT_FIRE_VOLLEY   indirect_fire_volley;/* Indirect fire volley  */
        INDIRECT_FIRE_REQ_PDU  indirect_fire_req;   /* Indirect fire request */
        TEMP_INDIRECT_FIRE     temp_indirect_fire;  /* Temp indirect fire    */
        BATTERY_FIRE_MISSION   battery_fire_mission;/* battery fire mission  */
    } variant;
} EAGLE_PDU;

/*** FUNCTIONS AND PROTOTYPES ***/
```

Note: Function and prototype definitions have been removed from this report for brevity.

```
/*** END OF FILE ***/
```

Appendix F : Technical Interchange Meeting 6 Trip Report

Traveler(s):	Clark R. Karr
Destination:	TRAC Facility in Fort Leavenworth Kansas
Dates:	August 4-5, 1993
Contract:	Integrated Eagle/BDS-D (64-12-313)
Purpose:	Initial Technical Interchange Meeting (TIM) for ECP1
Attendees:	TRAC: Jim Fox (LaRoque's replacement), Kent Pickett, Mike Hannon, Martha Moody LANL: Randy Michaelson, Deborah Kubicek

This trip achieved two goals. First, TRAC, IST, and Los Alamos National Lab (LANL) agreed on a preliminary workplan and schedule for the initial tasks in the Engineering Change Proposal (ECP1) of the Integrated Eagle/BDS-D contract.. Second, TRAC, IST, and LANL created preliminary designs for the interactions among the Eagle, Simulation Integration Unit, and the IST CGF Testbed systems for the first four tasks on the workplan. Additionally, the next three tasks on the workplan were discussed and preliminary ideas were considered. The attendees will consider these tasks in more detail in preparation for the next TIM scheduled for the week of October 4, 1993.

The first four tasks of ECP1 to be accomplished are: task 2 - Track Eagle Units in CGF Testbed, task 3 - Base Eagle Unit Detection on CGF Line of Sight, task 4 - Platoon and Battalion Disaggregation, and task 5 - Include Manned Simulators. These tasks were discussed in detail and several design issues resolved (see below for the details).

Draft copies of the IST prepared Integrated Eagle/BDS-D ECP1 Workplan and Interim Report #1 were presented as working documents for discussion.

The following is a summary of the discussions on each task:

Task 2 : Track Eagle Units in CGF Testbed

The following terms are defined:

Eagle world/region/area : the complete terrain used by the Eagle model.

Virtual world/region/area : the SIMNET or DIS terrain area. The virtual area is contained in the Eagle area.

Disaggregation region/area/box : an area within the virtual world where disaggregations can occur.

TRAC's responsibilities:

- Within the Eagle area, register the virtual area.
- For units with the virtual area, send Unit Status messages to the SIU.
  A Unit Status Message contains: unit identifier, alignment, echelon level, echelon type, heading, speed, operational activity, effectiveness, relation strength, remove unit flag, new unit flag.

LANL's responsibilities:

- SIU receives Unit Status Message.
- SIU produces Unit Appearance PDU (UAPDU) within IOP.
- SIU supplies unique Vehicle Number for each unit .
- When a unit is disaggregated, a UAPDU with the remove flag set is sent.
  UAPDUs are <u>not</u> sent while a unit is disaggregated.
- When a unit is reaggregated, SIU begins producing UAPDUs for the unit.

IST's responsibilities:

- CGF Manager receives UAPDUs.
- CGF Manager produces SIMNET Vehicle Appearance PDUs from data in UAPDUs.
- CGF Operator Interface is modified to display appropriate icon for unit.

Additional notes:

1. Adopt "echelon" for "unit" throughout code where sensible.
2. Put "start_vehicle_num_for_units" in loc_epro.h for SIU.
3. CGF Manager will not "time-out" a unit if UAPDUs not received for any length of time.

Definition of the UAPDU:

```
typedef struct
{
    unsigned long       vehicle;            /* Vehicle Id to be used in VAPDU.   */
    EAGLE_LOCATION      loc;                /* Location of the unit              */
    float               heading;            /* The units heading (radians)       */
    float               speed;              /* The unit speed (m/s)              */
    UCHAR               echelon_level;      /* Level of echelon                  */
                                            /*    (e.g. Company, Btln, Corps, etc.)*/
    UCHAR               echelon_type;       /* Echelon type                      */
                                            /*    (e.g. Infantry, Armored, etc.) */
    UCHAR               alignment;          /* Side of unit                      */
                                        /*(ENUM_EGL_ALIGNMENT_TYPES)            */
    UCHAR               effectiveness;      /* Current effectiveness             */
    unsigned short      relative_strength;  /* Percent active vehicles           */
    unsigned short      operation_activity; /* Operational Activity              */
    unsigned short      remove;             /* Remove unit from exercise         */
    char                unit_name[MAX_UNIT_NAME_LENGTH];
} UNIT_APPEARANCE;
```

Task 3 - Base Eagle Unit Detection on CGF LOS

TRAC's responsibilities:

- Eagle to issue Unit Status Report Request.

LANL's responsibilities:

- Collect CGF Sighting Report Messages
- Respond to Unit Status Report Request with Unit Status Report which contains a list of units and for each units the cumulative total of vehicles sighted.
- Evaluate compiler option to turn OFF byte alignment.

IST's responsibilities:

- CGF Testbed produces CGF Sighting Report Messages (defined in attached Fax).

Additional notes:

1. If compiler option (see LANL above) doesn't work, add padding fields in Sighting Report to align bytes.
2. Verify that vehicles can sight units and that the Testbed continues to function correctly.

Task 4 - Platoon and Battalion Disaggregation

TRAC's responsibilities:

- Provide to IST Battalion level Operational Activities (OA), Battalion template definitions, and mapping from OAs to templates.

LANL's responsibilities:

- Modify Disaggregation Response to include sequence and count fields.
- Modify Disaggregation Response handler to accept a series of Disaggregation Response PDUs for each disaggregation.

IST's responsibilities:

- Add sequence and count fields to Disaggregation Response PDU.
- Modify Disaggregation process to accept Battalion and Platoon disaggregations.
- Modify Disaggregation Response generator to produce a series of Disaggregation Response PDUs for each disaggregation.

Definition of the Disaggregation Response:

```
typedef struct
{
   UNIT_ID unit_id;       /* unit being disaggregated                    */
   UCHAR  success;        /* FALSE or TRUE                               */
   UCHAR  system_count;   /* Number of groups of entities in unit        */
   UCHAR  seq_length;     /* number of PDUs in this Disagg. Response     */
   UCHAR  seq_number;     /* sequence number of this PDU                 */
   UCHAR  data;           /* array[ systemCount ] of                     */
                          /*  DISAGG_RESP_SYSTEM_INFO                    */
} DISAGG_RESP;
```

Task 5 - Include Manned Simulators

TRAC's responsibilities:

- none

LANL's responsibilities:

- none

IST's responsibilities:

- Modify Disaggregation process to use manned simulators during disaggregation.

Additional notes:

1. IST's M1 simulator doesn't respond to Deactivation Request. Instead, the M1 will be beamed to location (0,0) and killed to deactivate it during aggregation.

Considerable discussion occurred covering the second set of three tasks (tasks 10, 14,& 19) in the IST Workplan. The following notes relate to this second set of tasks.

Task 14 - Indirect Fire at Eagle units

We assume the following: Eagle will disaggregate an artillery battery. Independent of Eagle, the human CGF operator will select targets for the battery and order the artillery battery to fire. Coordination of the disaggregated battery's fire with other Eagle artillery units will be a follow-on task.

TRAC's responsibilities:

- Send artillery battery disaggregation request.
- Receive Indirect Fire Volley (IFireVolley) and incorporate into Eagle's damage assessment system. Indirect Fire Volley contains: #rounds fired; center of volley; the width, depth, and orientation of volley sheath; and munition fired.

LANL's responsibilities:

- none

IST's responsibilities:

- Model battery fire behavior: IFire control in C^2 node.
- Produce IFire PDUs and IFireVolley.

Definition of the Indirect Fire Volley:

```
typedef struct
{
   EAGLE_LOCATION     target_loc;     /* Fire at location                */
   unsigned long      width;          /* Sheath width                    */
   unsigned long      depth;          /* Sheath depth                    */
   unsigned short     orientation;    /* Sheath orientation (degrees)    */
   unsigned short     rounds;         /* Number of rounds fired          */
   OBJECT_TYPE        munition;       /* Munition type being fired       */
} INDIRECT_FIRE_VOLLEY;
```

Additional notes:

1. Sending battery target list from Eagle to CGF a follow-on task.

Task 10 - Full Call for Fire

The purpose of this task is to enhance the existing, simple Call for Fire (Indirect Fire Request) to a request for indirect fire that supplies the information that Eagle requires for planning indirect fire. The model we are using is: the human operator is acting in a Forward Observer (FO) and requesting fire from Eagle.

TRAC's responsibilities:

- Define information in Call for Fire.

  To include: location
  desired effect of fire : from enumeration (suppress, neutralize, destroy)
  target echelon level : from enumeration (platoon, company, battalion, battery)
  target echelon type : from enumeration (tanks, apc, infantry, helicopter, command post, artillery, mortar, air defense)

terrain cover : from enumeration (open, covered)
vegetation : from enumeration (light, medium, dense)

- Perform target value analysis and allocate fire using Call for Fire.

LANL's responsibilities:

- Pass CGF Call for Fire to Eagle

IST's responsibilities:

- Implement in OI a facility to generate a Call for Fire request.

Definition of the Call for Fire PDU:

```
typedef struct
{
   EAGLE_LOCATION    target_loc;        /* Fire at location                  */
   UCHAR             effect;            /* Desired effect of fire            */
   UCHAR             target_ech_level;  /* Target echelon level e.g. company */
   UCHAR             target_ech_type;   /* Target echelon type e.g. armor    */
   UCHAR             cover;             /* Terrain cover                     */
   UCHAR             vegetation;        /* Vegetation around target          */
} CALL_FOR_FIRE;
```

Task 19 - Partial Disaggregation

There is difficulty in defining how this task is to be represented. There is a preference on the Eagle side for all or nothing disaggregations at the maneuver unit level. This task will be tabled until it is necessary to solve a problem in another task. This task may be required for task 16 - Patriot Trainer.

Task 15 - Direct Fire from CGF at Eagle Units

A possible scenario for this task incorporates task 16 - Patriot Trainer. A disaggregated Patriot launcher (the Patriot Trainer) attacks targets in a unit that is not disaggregated. The unit is described in UAPDUs and the individual vehicle positions (transmitted in Vehicle Appearance PDUs) come from templates in the CGF. This is a new class of unit: aggregate with VAPDUs being produced in the virtual region for its vehicles.

Other Issues:

The next Technical Interchange meeting is scheduled for the week of Oct. 4, 1993.

Send trip report with PDU definitions to Harry Jones by Wed. Aug. 11, 1993.

Ask Brian Goldiez if there are any administrative hurdles to bringing Scott Smith to IST? TRAC will determine how to house Mr. Smith. A Letter of Agreement to follow.

Appendix G : Technical Interchange Meeting 7 Trip Report

Traveler(s):	Clark R. Karr, Linda White, Eric Root
Destination:	TRAC Facility in Fort Leavenworth Kansas
Dates:	October 4-8, 1993
Contract:	Integrated Eagle/BDS-D (64-12-313)
Purpose:	Initial Technical Interchange Meeting (TIM) for ECP1
Attendees:	TRAC: Kent Pickett, Mike Hannon, Martha Moody
	LANL: Deborah Kubicek

This trip achieved two goals. First, TRAC, IST, and Los Alamos National Lab (LANL) installed and tested software for three of the first four tasks in Engineering Change Order 1(ECP1): task 2 - Track Eagle Units in CGF Testbed, task 3 - Base Eagle Unit Detection on CGF Line of Sight, and task 4 - Platoon and Battalion Disaggregation. Because there are no manned simulators at TRAC, task 5 - Include Manned Simulators was tested previously at IST. Second, preliminary designs and plans for the two tasks, task 14 - Indirect Fire at Eagle units and task 10 - Full Call for Fire, were discussed and agreed upon. As part of the discussion of the next phase of tasks, a extension to task 2 (Track Eagle Units in CGF Testbed) was discussed.

The extension to task 2 will allow individual vehicles of aggregate units to appear in the virtual environment in templated positions. Kent Pickett desires this extension to accomodate a studies of helicoptors as reconaissance vehicles and other sensor systems (e.g. JSTARS). In these studies, it is essential that the reconaissance vehicles and sensor platforms be presented with numerous individual vehicles in the environment. This can be accomplished by replacing the icons representing aggregate units with vehicles in templated formations. Clark Karr will coordinate with IST and STRICOM in Orlando to determine how to add this task to the list of deliverables.

The testing of the software went well with serveral minor software bugs being discovered. Test scenarios incorporating the following events were used:

1. Initialization of Eagle, SIU, and CGF system interconnection.
2. Eagle issues Unit Appearance PDUs have two blue and one red units.
3. Disaggregation of a Red unit.
4. Sending of Op Order for disaggregated Red unit.
5. Disaggregation of a Blue unit.
6. Sending Operator Intent messages from CGF OI.
7. Sending Indirect Fire Request from CGF OI.
8. Sending Frag Order from Eagle.
9. Request reaggregation from Red unit's CGF OI.
10. Reaggregation of Red unit.

The bugs discovered in the IST software were as follows:

Task 2:

1. Second and follow-on Unit Appearance PDUs didn't change the location and velocity of the unit icons on the OI. **(Fixed during testing).**
2. Echelon Vehicle Appearance PDUs have Site = 0 and Host = 0 instead of correct site and host. **(Fixed during testing).**
3. Red force unit icons have blue color. (To be fixed at IST).

Task 3:

no bugs found

Task 4:

no bugs found

Pre-existing functionality:

1. Indirect Fire Request PDU not sent from CGF Manager. (To be fixed at IST) All other Operator Intent message sent correctly.
2. Frag Orders (Advance Operation Orders) sent to new file rather than appended to existing Op Order file. (To be fixed at IST)

The following discusses the next two tasks and the extension to task 2.

Task 10 - Full Call for Fire

This task was reviewed and no changes from design at previous trip (TIM6)

Task 14 - Indirect Fire at Eagle units

We assume the following: Eagle will disaggregate an artillery battery. Independent of Eagle, the human CGF operator will select targets for the battery and order the artillery battery to fire. Coordination of the disaggregated battery's fire with other Eagle artillery units will be a follow-on task.

Two fields, unit_id and weapon_fired, need to be added to the INDIRECT_FIRE_VOLLEY. THe following is the new definition.

Definition of the Indirect Fire Volley:

```
typedef struct
{
   unsigned long        unit_id;        /* Firing unit's id                */
   OBJECT_TYPE          weapon_fired;   /* Weapon fired                    */
   OBJECT_TYPE          munition;       /* Munition type being fired       */
   EAGLE_LOCATION       target_loc;     /* Fire at location                */
   unsigned long        width;          /* Sheath width                    */
   unsigned long        depth;          /* Sheath depth                    */
   unsigned short       orientation;    /* Sheath orientation (degrees)    */
   unsigned short       rounds;         /* Number of rounds fired          */
} INDIRECT_FIRE_VOLLEY;
```

Clark Karr and Linda White had a discussion with Maj. Segres to get additional information about the command and control of an artillery battery. The following summarizes the discussion.

1. Fire Direction Officer (FDO) direct fire of battery (not battery CO).
2. Fire mission goes to the battery FDO who determines where the guns fire. Then the fire mission goes to the platoon and then to the individual guns.
3. Sequence of fire:
   a. individual guns fire all rounds as fast as possible. [note: 4 rounds per min. sustained rate]
   b. start of firing is either immediate upon receipt of fire mission or at command.
   c. Fire round at command is possible but little used.
4. Gun reports each round fired to platoon.
5. Platoon reports volley complete to battery.
6. Battery reports mission complete to Bn.
7. Two patterns of fire:
   point: 4 or 8 rds 50 meter radius from target
   area: sheath method.

Task 2 - Track Eagle Units in CGF Testbed (Extension)

An extension to this task to populate the virtual environment with larger numbers of vehicles was discussed. The goal of this extension to replace the single unit icon with vehicles in a templated pattern based on the unit's operational activity. This appears quite easy to accomplish. This task tracks and dead reckons Eagle aggregate units and issues single SIMNET Vehicle Appearance PDUs for eachunit every 5 seconds as the units move around the battlefield. The extension will issue Vehicle Appearance PDUs for individual vehicles in a templated formation around the units' centers of mass. Note that only the unit is dead reckoned and that the individual vehicles are not simulated in any fashion; this will minimize the overhead of putting lots of vehicles in the environment.

The following is the definition of the DETAILED_UNIT_APPEARANCE PDU (DUAPDU) that Eagle will send to the CGF Manger every time step.

Definition of the DUAPDU:

```
typedef struct
{
   DETAILED_UNIT_INFO  unit_info;      /* Specifications for disaggregation   */
   UCHAR               system_count;   /* Number of groups of entities in unit */
   char                padding[3];     /* Forces data to 4 byte boundary      */
   char                data;           /* Array of 'systemCount' elements of  */
                                       /* System Info.                        */
} DETAILED_UNIT_APPEARANCE;

typedef struct
{
   unsigned long       start_vehicle_num; /* Vehicle num to be used in VAPDUs*/
   EAGLE_LOCATION      loc;            /* Location of the unit                */
   float               heading;        /* The units heading (degrees)         */
   float               orientation;    /* The units orientation (degrees)     */
   float               speed;          /* The unit speed (m/s)                */
   UCHAR               echelon_level;  /* Level of echelon                    */
                                       /*    (e.g. Company, Btln, Corps, etc.)*/
   UCHAR               echelon_type;   /* Echelon type                        */
                                       /*    (e.g. Infantry, Armored, etc.)   */
   UCHAR               alignment;      /* Side of unit                        */
                                       /*(ENUM_EGL_ALIGNMENT_TYPES)          */
   UCHAR               effectiveness;  /* Current effectiveness               */
   unsigned short      operation_activity; /* Operational Activity            */
   unsigned short      remove;         /* Remove unit from exercise           */
   char                unit_name[MAX_UNIT_NAME_LENGTH ==20];
   char                system_count;   /* Number of groups of entities in unit */
   char                padding[2];     /* Force alignment to 4 byte boundary*/
   char                data;           /* Array of 'systemCount' elements of  */
                                       /* System Info.                        */

} DETAILED_UNIT_INFO;
{crk - consider changing heading and orientation to shorts for space}

/* This is not a new structure; it exists already in loc_epro.h.  It is included here for clarity*/
typedef struct
```

```
{
   ULONG          entity_type;       /* DI, ABRAMS, BMP, etc.              */
   int            num_of_entities;   /* Number of this type in the unit    */
   char           padding[2];        /* Keeps structure on 4 byte boundary */
} SYSTEM_INFO;
```

One of the issues we will need to resolve is how to keep vehicles from floating or sinking without frequent dynamics checks. We should consider checking the normal of polygon under vehicle and sending vehicle along surface of the polygon. This will minimize floating and sinking but not eliminate it because the vehicle will float or sink when it leaves the polygon. Experimentation will reveal an acceptable rate of recalculating the vehicle attitude relative to the underlying polygon. Note that the slower the unit is moving the less the problem (stationary - no problem).

Preliminary Schedule

We agreed on the following schedule:

Oct. 15 : trip report and Detailed Unit Appearance PDU designed and sent to Ft. Leavenworth.
Oct. 22 : all bugs fixed and sent to Mike Hannon.
Nov. 15 : potential Technical Interchange Meeting in Orlando
Dec. 6 : potential trip to Boston to evaluate ModSAF.
This trip is not funded in the current contract.

Additional notes:

- The SAFDI bug fix which increments vehicle # in Vehicle Appearance PDUS in not completely implemented. The OI shows non-incremented vehicle #s and the Sighting Reports use non-incremented vehicle #s. We should consider either:
  a) consistently implementing the incrementing approach (the vehicle # the OI Name Display option shows needs to be determined),
  b) discard the 0th array element and use 1 through n for vehicle #s, or
  c) put a Manager in position 0 and use 1 through n for vehicles #s.

- Add "element" to list of echelon levels.
- Add "CEWI" and "TANK_ONLY" to disagg echelon types.
- Use degrees; N == 0°, clockwise rotation.