Automated Second Echelon Commitments for the C3EVAL Model

David Warren Roberson

TR 88-37

ABSTRACT

The C3Eval model is being developed by the Institute for Defense Analyses (IDA) for the Joint Chiefs of Staff (JCS/J6). The model is to provide one element of a workstation being developed for JCS that is to be used to evaluate theater level command, control and communications (C3) in terms of combat consequences. The red commander enhancement to the C3Eval model provides a red side that is independent of the blue side. The enhancement also automates the commitment of second echelon forces on the red side. This research report describes the foundation for this enhancement.

CR Categories and Subject Descriptors: I.6.3 [**Simulation and Modeling**]: Applications; H.4.2 [**Information System Applications**]: Logistics.

General Terms: Management, performance, measurement.

Additional Key Words and Phrases: Command, control and communications; theater level; model enhancement.

Technical Report SRC-88-007

AUTOMATED SECOND ECHELON COMMITMENTS FOR THE C3EVAL MODEL

David Warren Roberson

Systems Research Center
Virginia Tech
Blacksburg, VA 24061

May 1988

This research was supported by Applications Research Corporation, Falls Church, Virginia through the Systems Research Center at Virginia Tech.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
Background and Overview

1.1 General subject of C3

1.1.1 What is C3?

In order to discuss intelligently a model of command, control and communications (C3) one needs to have a clear understanding of what C3 is in the real world (the system). An in-depth understanding of the system is a prerequisite for selection of the most important elements to be simulated.

C3 represents the key elements of the management structure of the military. The conceptual framework of the military can be very closely paralleled with that of a large, free-enterprise business corporation. The purpose of, or the motivating factor behind, a company is to make a profit. This can be thought of as attempting to beat a rival company at acquiring resources. The same concept can be applied to a military force.

An obvious distinction within a company is made between technical (line) and managerial (staff) functions. The technical functions are those tasks which generate the product of the company. The managerial functions are the behind the scenes tasks that keep the technical side operating. The management side is the master controller for

all the individual elements of the technical side. The management side has a pyramidal hierarchy which allows for aggregation of the information into reports as the information flows towards the peak of the pyramid. This reduces the amount of data received at upper levels of the hierarchy. This reduction is necessary so that the upper levels of management can process the incoming reports in an acceptable period of time and create an overall picture of the company's status.

This hierarchical structure consists of three major elements. The first element is the decision makers. These decision makers are the nodes defining the basic (skeletal) structure of the pyramid. The links between the nodes represent the flow of information between the nodes. This flow of information is the second element of the structure. The third element is the control exercised by each node in the structure. In a strictly hierarchical organization a given node controls all the nodes that are located in the structure directly below it. Generally, the lines of control follow the lines of communication.

This characterization or model of a company can be easily extended into the military realm. In the military the same division occurs. The technical side consists of the combat units and C3 represents the managerial side. Once again the function of the management element is to choose

the overall direction the company is to take and to coordinate the interaction of the individual elements on the technical side. In this way management determines the overall plan of action and assigns elements from the technical side to reach the goals required by the plan of action. The military management has a strictly pyramidal hierarchy and many of the same inherent properties as a company.

This work utilizes the basic hierarchical structure described above for characterizing command/ control/ communications (C3):

(1) The nodes of the structure represent the command portion,

(2) The flow of information between the nodes is the communications portion, and the topological structure defines the control portion.

1.1.2 Why is study of C3 important?

Why spend the money and effort to understand and improve the C3 realm of the military? The most obvious answer is that like any large corporation, the military cannot perform at its maximum efficiency without good management. The need for rapid communications and decision making is driven by the decreasing time for reaction afforded to commanders. The decreasing time for reaction is caused by technical advances in the speed and mobility of

our foes. The decrease in the time needed to react is made possible by advancement in communications and automated decision aids. Currently, information must be transmitted, processed and acted upon in far less time than ever before [Herres 1987].

An amplifying factor to the decreasing reaction time is the lack of education or training in the fundamental concepts of command and control. Accordingly, users as well as researchers have not reached the desired levels of expertise. In addition, these two groups are unaware of each other's limitations. The private contractors who develop the C3 software (the researchers) do not fully understand the needs of the military personnel using the software (the users). Also the users do not understand the limitations faced by the researchers. Needed are standard methods and techniques within the C3 realm. Models of C3 are needed for both understanding & teaching [Herres, 1987].

1.1.3 What alternative models are there to C3Eval?

Currently, models of both C3 and combat exist, but no models possess bidirectional interactions between explicit representations of both components. Most models of combat use Lanchester's equations. Lanchester derived several laws that described various forms of confrontation. These

various laws covered items from single man-on-man engagements to area fire (or invisible firing) to modern fighting with long-range weapons and the concentration of superior numbers.

The models currently being developed implement bidirectional interactions between C3 and combat. One such model is being developed at the Naval Postgraduate School using "modern methods of nonlinear nonequilibrium statistical mechanics, to develop probabilistic models of combat scenarios derived from combat simulations" [Ingber, 1987]. Basically, their approach is to create a mesoscopic scale between the microscopic scale (detailed individual interactions) and the macroscopic scale (command level of aggregated information). This mesoscopic scale is the C3 interface between the two levels. Using this method, they derive nonlinear multivariate functions describing drifts (trends) and diffusions (risks). The output of this system is probability distributions of combat-relevant variables which provide the most likely states of the system. The approach used is a deviation from Lanchester's theory [Ingber, 1987].

The Conflict Model (ConMod) is another model being developed to respond to similiar needs. However, based on a control system approach to organizations, the approach of ConMod is quite different. The model separates the world

into three parts, one physical plane and two cognitive planes. Both the red and blue sides have their own cognitive planes but they share the one physical plane. All interaction between the two sides occurs within this shared plane. ConMod uses an object oriented approach to represent the military objects and command and control (C2) networks. Objects interact by causing and feeling the effects of various actions. These actions are scheduled as cause and effect events. In this way, ConMod uses discrete event simulation. The physical actions in ConMod are modeled by three-dimensional digitized terrain with resolution capability to represent single items as well as systems.

1.2 "Workstation Concept" with C3Eval as one element

Another project being designed to enhance the military's capability in the C3 realm is a work station for the commander's in chief. Current analysis capabilities are limited by the complexity in locating, retrieving and integrating data residing in multiple data base management systems. This is further complicated by data residing across different computers and operating systems. The development of a unified information concept to support the analytical functions is enabled providing location, acceptance and integration of elements, subsystems and

interfaces. This concept can be considered as a work station which provides:

(1) A consistent "user-friendly" interface to data in multiple data sets, DBMS's, and hardware sites.
(2) Identical appearance and function at and across different computers and operating systems.
(3) A uniquely flexible interface to create disciplinary functions from engineering process modules.
(4) Functional distribution and integration.
(5) Translation of data elements and processes into scientific and engineering information.
(6) Continued operation with non-available elements that are duplicated elsewhere.
(7) Operation on improved hardware.

This proposed work station has six functions: data management, data aggregation, evaluation of C3, analysis aids, displays, and document generation. The third function is to be performed by the C3Eval model.

1.3 Overview of C3Eval model

1.3.1 Purpose of the model

The C3Eval model is being developed by the Institute for Defense Analyses for the Joint Chiefs of Staff (JCS/J6). The model is currently targeted for IBM AT's, IBM XT's, VAX 785's, and Micro VAX's. The purpose of the model is to analyze C3 capability at the theater level.

1.3.2 Model's approach to measuring C3

The effectiveness of communications can be measured directly by the effectiveness of the communications

hardware. However, the measurement of the effectiveness of command and control (C2) cannot be measured in this way. "C2 effectiveness can only be addressed directly by requiring that all C2 actions be judged by discernible combat outcomes" [Robinson, 1987]. In order to achieve this, the model must take a closed loop approach towards C3 and combat. This is achieved by separating the model into two distinct modules, the first module performs the C3 and the second module performs the combat. In order to achieve the closed loop this cyclical sequence must be followed:

(1) Combat results reported to C3 side.
(2) Messages processed and acted upon by C3 side.
(3) Traceable messages sent to combat side.
(4) Simulated change in combat observed and "captured".

1.3.3 Model's representation of C3

The C3 module represents the communications paths between two command units (or nodes) as aggregations of each of the types of communications links that exist between the two nodes. The information flow between nodes is represented by explicit messages that are sent and received by explicit nodes. The C2 processes are represented by (1) decision making based on information located at the node pertaining to requests for combat support, and (2) messages generated by the decisions made in (1) [Robinson, 1987]. Each new message is generated according to the output format

of the rule performing the process. Rules are either periodic (the appropriate time interval has elapsed since the previous activation of the rule) or aperiodic (message arrival triggers the processing of the rule).

1.3.4 Key elements of the model

The model is time-stepped (currently, time increments of one-half hour) and scenario-driven; critical events that occur during the time of simulation must be predetermined and logged in a data file for access by the model. This data is stored as external messages introduced into the communications network at the time specified by the creation time attribute of the messages. The input to the model also contains the data sets for the nodes, the communications links and the rules. The rule data set contains the rules that determine (1) the input required at a node to perform a particular process and (2) the format of the output messages generated by that process. These rules form the basis for the flow of information (messages) between nodes and the processing of the information received at a node. The creation of the input files is made easier and more user friendly by the preprocessor, a menu driven, data dictionary based, and form designed interface with the user.

There are three distinct elements simulated by C3Eval. The first is the C3 environment. It consists of a set of

nodes (command posts), paths (lines of communication) and processing of messages and combat data. Each combat level node has specific ground-based weapon systems (for example: tanks, anti-aircraft artillery, and infantry). In addition, a notional airbase has aircraft (Blue only) which are requested by "air tasking order" (ATO) messages for specific combat units at a specifed game time interval. These weapon systems and aircraft sorties are the second element. The third element of the model is the combat process. The combat drawdown is calculated using a matrix method developed by IDA [Anderson, 1984]. Currently the combat portion of the model has an explicit blue side with implicit red opponents.

The execution of the model consists of repeatedly executing a well defined loop. This loop is executed once for each time step. The sequence of the loop is:

(1) Limit input messages.
(2) Process messages received.
(3) Process requests for combat support.
(4) Create messages based on command post actions.
(5) Limit output messages.
(6) Allocate messages to network.
(7) Put messages into communication status.
(8) Schedule and deliver supplies.
(9) Process requests for CAS.
(10) Update combat unit's weapon systems.
(11) Calculate combat drawdown.

Model output is placed in two types of output files. The first file type allows the user to review details of the

simulation. These files contain all output generated during each time step and summary reports. The output generated during each time step includes the tracing of all messages flowing between nodes as well as status reports for each unit in combat. The second file type is the graphics data that contains summary information about message traffic and casualties at each node (unit). A graphics postprocessor uses this second type of file to make graphs.

1.3.5 Present and future uses of the model

Currently, the assessment of C3 effectiveness measures the degradation of communications as well as the degradation of command posts over a period of several days. The effect of increased communication capability as well as increased command post efficiency can be evaluated. New C3 systems can also be evaluated in terms of their impact on operations.

The model could also be modified to perform several other functions. The model could be used as a man-in-the-loop teaching tool. This would allow new officers a chance to interact with a simulated environment so that they could see the results of decisions they made. The model could also be used for evaluating naval combat outcomes as a result of C3 capabilities.

1.4 Organization of this report

This chapter is organized to reflect the subject relationships depicted by the three largest (outermost) circles in Figure 1. The discussion begins with background information on the field of C3, then narrows its area of concern to the proposed conceptual JCS Workstation. The chapter concludes with an overview of the C3Eval model which is the element of the workstation serving as the context for the work reported herein.

Chapter 2 continues the discussion of the C3Eval model and describes the red commander enhancement. This chapter also motivates the need for the red commander enhancement.

Chapter 3 gives the manager's/systems analyst's view of the work done on the red commander enhancement. This chapter consists of two sections: the first section explains the organization of the effort whereas the second section describes the specific approaches used to develop the enhancement. This chapter emphasizes the research project portion of the red commander enhancement.

Chapter 4 gives the programmer's view of the representational requirements for the enhancement. Chapter 5 describes the test suite used to validate the code generated by the research project. Chapter 6 is a summary/critique of the research project and of the C3Eval model.

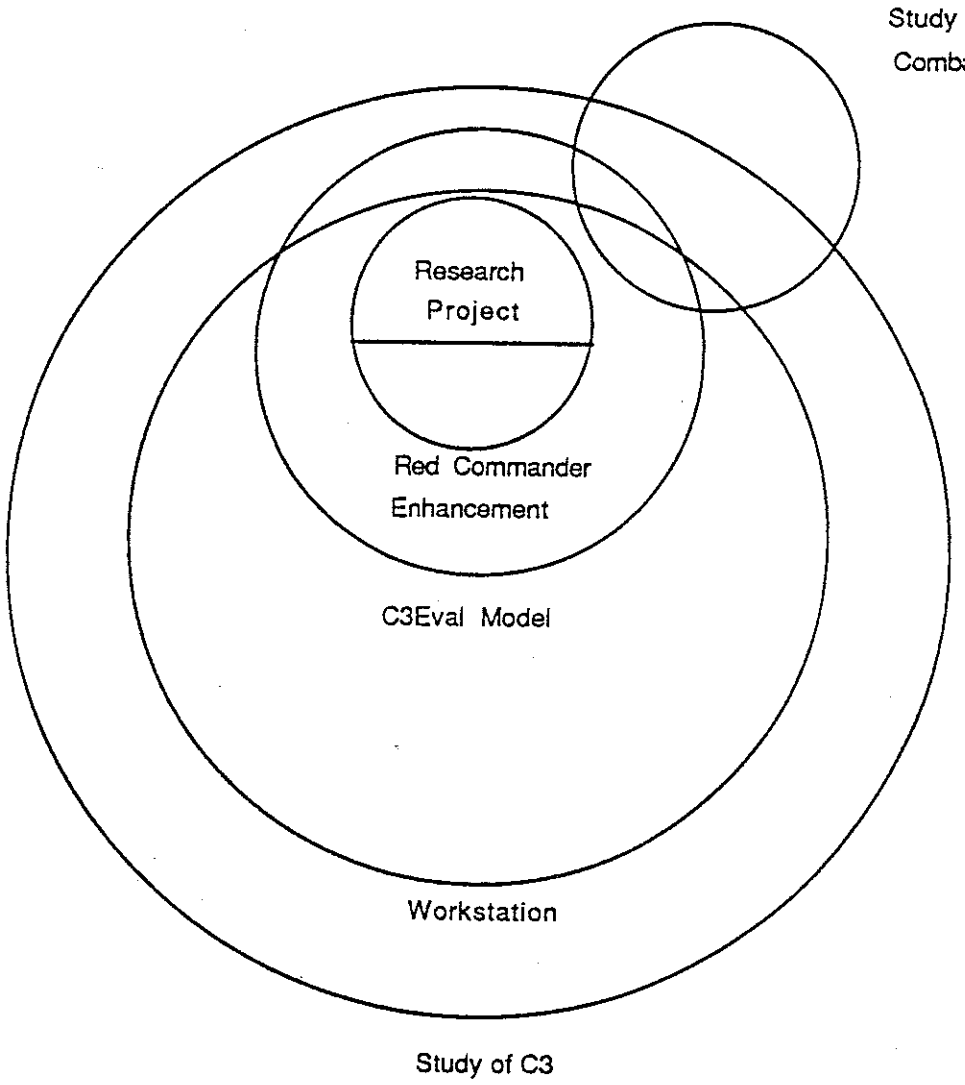


Figure 1 Venn Diagram of Subject Relationship

CHAPTER 2
Research Project Objectives

2.1 Description of the red commander enhancement

The red commander enhancement to the C3Eval model embodies the explicit representation of the commitment of red units. This research project is only a part of the red commander enhancement. The enhancement to the C3Eval model requires modifications of the model beyond those made during this project.

The explicit representation of the commitment of red units is accomplished in two levels of implementation. The first level, which is performed by the code generated by this research project, is the division (or combat unit) level commitment of red forces. At this level the subordinates are implied. The second level entails the modifications required beyond the scope of this research project: the Red Army Group Commander (or intermediate commander) level commitment of red forces. This level has actual subordinates. Both levels require the ability to utilize perceptions of combat foes and subordinates. For the relationship between these levels see Figure 2.

The second echelon commitment algorithm is an expert system that uses input parameters to determine the effect of its decision rules. The expert system is heuristic, but has

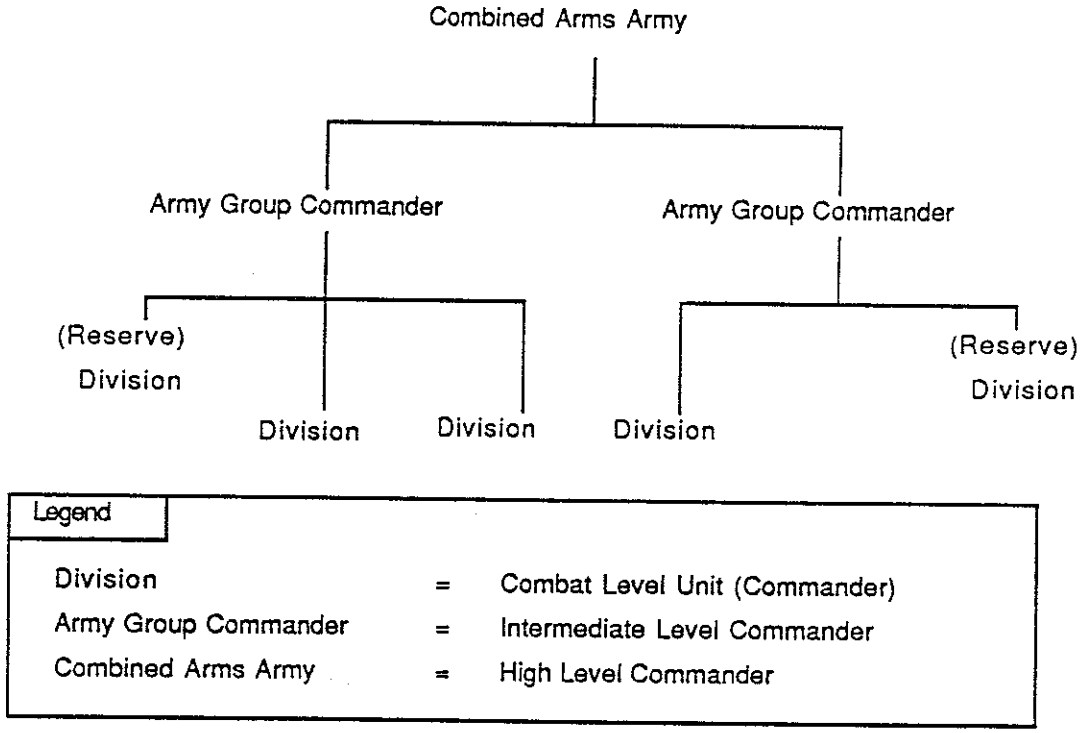


Figure 2 Example hierarchy showing the three levels for the red commander enhancement

an alternative stochastic mode of operation. The action taken as a result of the command post decision differs for the army and division levels. The army level has the ability to send communications of orders to subordinates. The division level has the ability to control the level of combat intensity.

At the division level (developed by this research project) there are three factors which control the level of combat intensity. The main factor is the goal force ratio which must be met by the goal time. The two other factors are the low and high bounds for the unit's force ratio. Since a unit must stay within its bounds, even after reaching its goal force ratio, the model has controlled combat. Consequently, the model cannot provoke an unrealistic attrition of the opponent's forces.

2.2 Impact on the C3Eval model

The red commander enhancement impacts at three levels on the red side (Figure 2). The first level is at the combat unit level. At this level the unit has one function to perform, which is to meet and sustain force ratio goals. This is accomplished by controlling the unit's combat operations status matrix, posture, engagement rate, and reserves. The combat unit level representation of impact is the primary objective of this research project.

The second level of impact is at the intermediate commander level (Army Group Commander). The Army Group Commander has two functions: (1) review actions of each subordinate combat unit and if appropriate, replace combat unit commander; and (2) attempt to meet and sustain its own goals. The high level commander (Combined Arms Army) determines the intermediate commander's goals. These goals are met by changing subordinate combat unit's goals and committing second echelon units. The first function is closely related to the impact at the first level and therefore is accomplished by this research project. The second function is an extension of the effort planned to follow this research project.

The third level of impact is at the high level commander (Combined Arms Army). The high level unit has one function; to initiate the red force's concept of operation by issuing plans to the intermediate commanders and by checking the success of those commanders in meeting their plans. This function is closely tied with the second function of the intermediate commander. The high level commander function is also an extension that is dependent on the efforts in this research.

2.3 Benefits of the red commander enhancement

There are four major gains from the explicit red commander addition to the C3Eval model. First, the addition provides the ability to evaluate the first order impact of red C3 on combat outcome. The second result is the ability of the blue side to counter red C3 operations. Third, the red side representation emphasizes the blue unit actions in the prescripted scenario (input of actions by players). Finally, the model might be employed as a teaching tool. In order to use the model in this way the model must be redesigned to permit a "man-in-the-loop" capability. Such a redesign mandates a responsive red side so that the competitive aspect can promote the learning potential of the model.

CHAPTER 3
Research Project Approach

3.1 Management of the research project

This project consists of five activities. The first activity is the design phase, which includes the development of the algorithms to be used. Reading of background material and technical discussions with employees of ARC and IDA are the key elements. The second activity is the development of the detailed design. The third activity is the coding of the detailed design and the testing of the code. Although considered a fourth activity, the documentation phase is in concert with the coding. The fifth activity is the rehosting of the code at IDA. The details of the schedule are shown in the Schedule and Milestones (Appendix B).

The larger model development effort employs two teams working simultaneously on independent but related tasks. The first team consists of employees from the Washington, D.C. branch of ARC. The author, assisted by the supervising professor, is the second team. This allowed each task to be tested independently. There also existed a dependence between the tasks for the two teams. The three tasks essential to this project and their relation to the larger development effort are shown in Figure 3.

Team one tasks

0) Create corps level data base.

1) Create red side with message capability.

2) Create red combat unit perceptions.

3) Create red intermediate commander perceptions.

4) Modify combat portion of the model.

Team two tasks
(Research project)

1') a. Allow a combat unit or intermediate level commander to log a plan for himself.
b. Allow an intermediate level commander to log plans for subordinate units.
c. Allow a commander to handle multiple "plan" messages being received during one time increment.

2') Allow a combat unit commander to evaluate and react to his status with respect to his plan.

3') a. Allow an intermediate level commander to evaluate and replace a combat unit commander.
b. Allow a combat unit to replace its commander.

NOTE: (1) Team one is employees from the Washington, D.C. office of ARC.

(2) Team two is the author assisted by the supervising professor.

(3) Task numbers show synchronization points between the two teams.

Figure 3
Task relationship

3.2 Top level design

3.2.1 Design techniques and design considerations

The understanding of several techniques used in the current model is necessary to an explanation of the algorithms developed during the research project. First, the communication portion of the model is represented by explicit messages rather than using statistical methods. Second, each node (unit) has its own rules which are used to trigger processing of the unit's input messages and to specify the format of the output messages. Third, all of the data for each node and for the communications network is stored in dynamically allocated virtual memory space. This memory space is used with the basic pointer system concept. The size and contents of the blocks (or records) contained in this memory space is described by Dynamic Data Structure (DDS) blocks. Examples of these DDS blocks are seen in Appendix C.

3.2.2 Algorithms developed during project.

The algorithms used to perform the red commander decision making are developed during the design phase. Technical discussions with IDA personnel represent the sources for the formulation of these algorithms. A combat unit commander follows a plan of action dictated to him by his intermediate level commander. This plan of action

specifies a goal force ratio that the unit is to reach by the specified goal time. At goal time minus combat reaction time, the unit checks its goal status (has the goal been achieved). If the unit has not reached its goal, the unit takes the appropriate action to increase its force ratio value. (Combat reaction time is a constant increment (delta) specified by the user.) The unit is also given low and high bounds for its force ratio. If the unit crosses either of the force ratio bounds then the unit takes the appropriate actions and sends a message to its intermediate level commander specifying the action taken.

The intermediate level commander has three tasks to perform and does these based on perceptions. The first task is to check if each subordinate combat unit has reached its goal force ratio by its goal time. If a unit does not reach its goal, then the combat unit's commander is replaced. The second task is to check a unit's action; if the unit acted inappropriately then the combat unit's commander is removed. The third task is to perform periodic spot checks of the subordinate combat units; if a unit performed inappropriately then the combat unit's commander is removed.

From Figure 3, the incorporation of these algorithms into the existing C3Eval model is divided into three tasks.

Task 1' added the ability of a unit to receive "plan"

messages, to process the messages, and to send "plan" messages to its appropriate subordinate units. This task is further divided into three logical elements, each element being an expansion to or modification of the previous elements.

The first element gives a unit the ability to receive and log a "plan" message containing one plan for the unit. This requires the addition of two rules to the rule data base. The first rule (number 4520) allows the intermediate level commanders to receive and process a "plan" message sent from a high level commander. Currently the only way for this message to be generated is by external message. The second rule (number 3020) allows the combat level commanders to receive and process a "plan" message sent from an intermediate level commander. The first element of task 1' also requires the addition of a DDS block which contains the data for the plan. Finally, the code to process the incoming "plan" messages is developed. This code is called whenever either rule number 4520 or rule number 3020 is activated.

The second element of task 1' expanded a unit's capability to receive and process a "plan" message containing a list of plans. The list could contain a plan for the unit as well as a plan for each of the unit's subordinates. This requires the modification of the code

written in the first element of task 1'. The code has to handle a list of plans instead of a single plan. The code also has to perform additional processing for all the subordinate unit plans. These plans must be saved with the unit's data for the appropriate subordinate unit. Each of these plans also has to be sent in a "plan" message to the appropriate subordinate unit.

The third element gives a unit the ability to process more than one "plan" message during one time increment. This requires modifications to the code written during the first two elements of task 1'. The modification is a routine that takes the list of "plan" messages received and merges all the plans contained within the messages into one master list of plans. In this way the incoming data is maintained in the correct form to interact with previously written code.

Task 2' adds the ability of a combat unit to evaluate its status with respect to its plan and to take the appropriate action necessary to change its status. This requires the addition of the periodic rule (number 3024) which allows a combat unit commander to evaluate his status and if appropriate send a message to his intermediate level commander. This also requires the development of the combat operations status matrix. This matrix is used by the combat

unit for a status change to correct a deviation from the unit's plan. This also requires the development of the code to evaluate the unit's status and to send the message, if appropriate. This code is called whenever rule number 3024 is activated.

Task 3' is divided into two elements. The first element adds the capability of an intermediate level commander to evaluate the action taken by a combat unit commander. If the action taken is inappropriate, then the combat unit commander is replaced. This requires the addition of a rule (number 4521) to allow the intermediate level commander to receive "reaction" messages from subordinate units and to send "change of commander" messages to subordinate units. This also required the writing of code to evaluate, from perceptions, the action taken by the combat unit commander. This code is called whenever rule number 4521 is activated.

The second element of task 3' adds the ability of a combat unit to receive a "change of commander" message. This requires the addition of a rule (number 3026) to allow the combat unit to receive a "change of commander" message. This also requires the writing of the code to process the message. This processing consists of changing the unit's combat operations status index, posture and engagement rate. This code is called whenever rule number 3026 is activated.

CHAPTER 4
Research Project Implementation

4.1 Detailed Design

As seen in Chapter 3, the project consists of three tasks. The first task allows red units to receive, to process and possibly to forward "plan" messages. The second task allows red combat unit commanders to evaluate and to react to their status with respect to their plans. The third task contains two functions. The first function is to allow red intermediate level commanders to evaluate the actions taken by subordinate commanders and replace them if appropriate. The second function is the actual replacement of a combat unit commander. This totals to four functions. Each of these functions is called whenever the appropriate rule is activated. The activation of a rule occurs whenever a node receives a message whose type is specified as the required input for the rule in question. The output format of the rule is used to format any output generated by the function called to process the message(s).

The hierarchy chart shows the four new functions (Figure 4). The large majority of the existing code is not shown. This chart shows that subroutine Process is the routine that calls the functions that are invoked when a rule is activated.

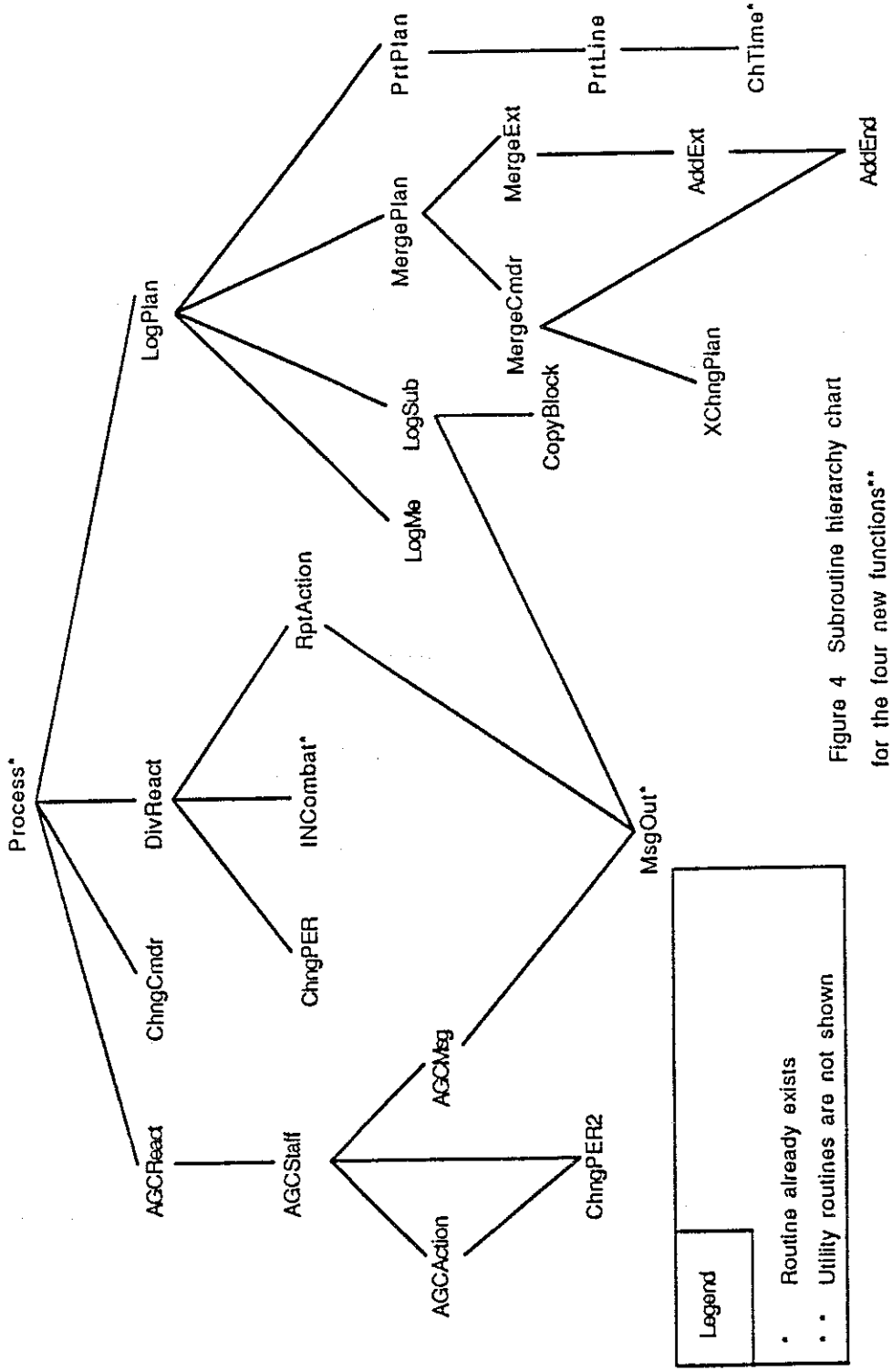
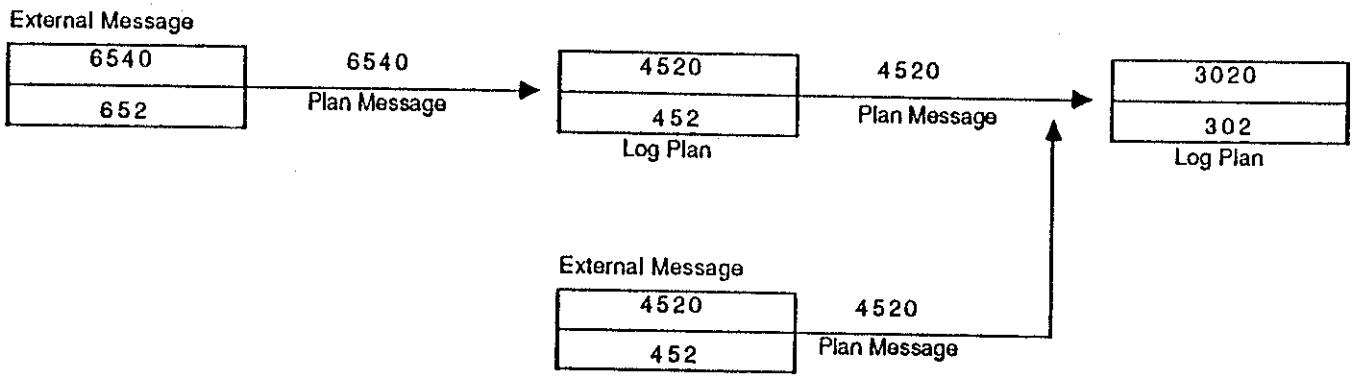


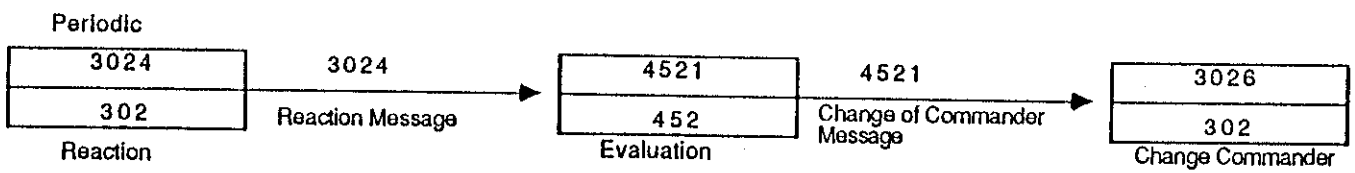
Figure 4 Subroutine hierarchy chart for the four new functions**

The hierarchy chart shows the relationship among the subroutines written to perform the four functions. Figure 5 denotes the relationship between the rules that activate these functions. Each box in the figure represents a rule (except boxes labeled with External Message). The top number in the box is the rule number and the bottom number is the number of the associated node (unit). The arrows directed into a box represent the type of message being received at the node, and the arrows directed out represent the type of message(s) being generated by the node. A rule is activated whenever a node of the appropriate type receives an input message of the specified type. A rule can also be activated by time (in the case of a periodic rule). The boxes labeled with "External Message" represent external messages which appear in the communications network at the time specified by the user. These messages are identical to the messages created by the rule indicated within the box except that the identifier of the originating unit is set to identify the message as externally generated.

Figure 5(a) shows the flow of red "plan" messages among the high level, the intermediate level and the combat unit level commanders. Note that at the intermediate level a "plan" message can only be received externally. This is because currently the high level commander node on the red side is non-existent.



(a) Red Plan



(b) Red division action and reprisals

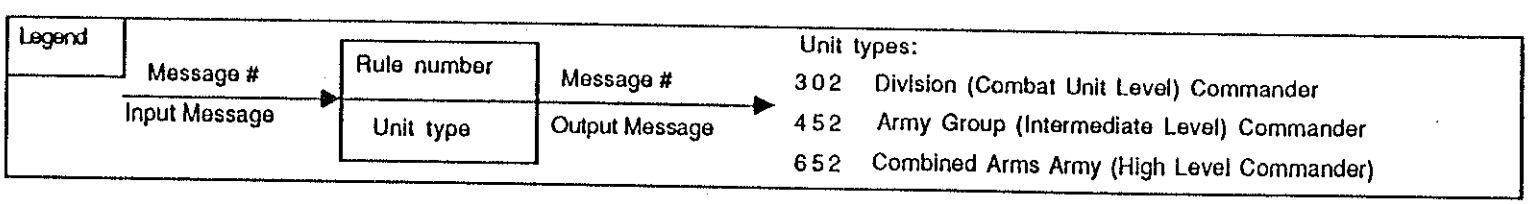


Figure 5 Rule Interaction Chart

Figure 5(b) shows the flow of messages for combat unit level commander actions and intermediate level commander reprisals. Note that rule 3024 is periodic which means it is activated by time and not by the event of a message arrival.

The next four sections of this chapter describe the four functions in further detail. Each section describes the rule or rules developed to perform the function being described. Each section also describes the dynamic data structure blocks developed or modified. The actual DDS blocks are found in Appendix C. Next, each section discusses the initialization required (or any other special requirements) and the flow of control among the subroutines that perform the desired function. Finally, each section contains a description of each of the subroutines. The top level subroutine (for the function in question) is described first. The remainder of the subroutines for that function are described in alphabetical order thereafter.

4.2 Function one: Logging plans

The ability to log a plan(s) is achieved by rules 4520 and 3020. Rule 4520 allows intermediate level commanders (nodes of type 452) to receive "plan" messages (messages of type 6540) and if necessary send "plan" messages (messages of type 4520) to subordinate units. Rule 3020 allows combat

unit level commanders (nodes of type 302) to receive "plan" messages (messages of type 4520). The activation of either rule results in the calling of subroutine LogPlan.

Two DDS blocks have been created and two blocks have been modified. The new DDS block labeled PLAN contains the data for a unit's plan. The new block labeled MSGIN is used as the header for a list of PLAN blocks sent in a "plan" message. The two existing blocks have been modified so they each have a pointer to a plan block. The NODE block contains a pointer to the plan for the unit identified by the NODE block. The STATUS block contains a pointer to the plan for the unit identified by the STATUS block. The list of STATUS blocks contains one entry for each subordinate of the unit for which the list exists. Therefore, a unit has a plan for itself which is accessed from the NODE block and the unit has a plan for each of its subordinates which is accessed from the STATUS block for each of the subordinates.

Currently, the only way to initiate "plan" messages is by external message. Therefore, the ability to accept (through input) external "plan" messages is required. The existing driver routine for the input of external messages, subroutine ExtMsg, calls subroutine ReadPlan for every external "plan" message accepted for which additional data lines are indicated. Subroutine ReadPlan reads each additional data line and creates an extension to the "plan"

message containing the contents of the data line. ReadPlan then returns control of execution to the driver routine.

The top level routine for processing plans is LogPlan. Subroutine LogPlan first calls subroutine MergePlan to merge all the plans contained in the "plan" messages received into one master list of plans. Then LogPlan repeatedly calls either subroutine LogMe or subroutine LogSub to process each of the plans in the master list. Subroutine LogMe is used to log a plan for the unit performing the process. Subroutine LogSub is used to process a plan for a subordinate of the unit performing the process. This processing consists of logging the plan and sending the plan in a message to the subordinate unit. After subroutine LogPlan has processed all the plans in the master list, LogPlan calls subroutine PrtPlan to print out the updated plan for the unit performing the process.

4.2.1 Subroutine LogPlan

Subroutine LogPlan allows a red unit to process any "plan" messages received during one time increment. Each message can contain a list of plans. This list can have a plan for the unit processing the message as well as a plan for each of the unit's subordinates. The input parameters for this routine are the pointer to the unit performing this

process, the pointer to the output message process structure which contains the parameters for message handling and the pointer to the output message format.

Three tasks must be performed by subroutine LogPlan. The first task is to merge all of the plans contained within the messages received into one master list of plans. This task is performed by subroutine MergePlan. The second task is to log each plan in the appropriate location. If the plan is for the unit processing the message, then subroutine LogMe is used to log the plan. If the plan is for a subordinate, then subroutine LogSub is used to log the plan and to transmit the plan to the appropriate subordinate unit via a "plan" message. The third task is to print out the updated plans that the unit has for itself as well as for its subordinates.

4.2.2 Subroutine AddEnd

Subroutine AddEnd adds a plan at the end of a list of plans. The input parameters are the pointer to the plan to be added to the list, the pointer to the top of the list and the pointer to the bottom of the list. The output parameters are the updated pointers to the top and bottom of the list.

4.2.3 Subroutine AddExt

Subroutine AddExt adds the plans contained within an external message to the master list of plans. The input parameters are the identifier of the node processing the message, the pointer to the external message containing the list of plans to be added to the master list of plans, the pointer to the top of the master list of plans, and the pointer to the bottom of the master list of plans. If a plan contained within the message is for a unit that already has a plan in the current master list then (1) do not add the new plan to the master list and (2) send an error message to the user. Otherwise, call subroutine AddEnd to add the plan to the end of the master list. The plans in the master list are not kept in any ordered sequence. The output parameters are the updated pointers to the top and bottom of the master list of plans.

4.2.4 Subroutine CopyBlock

Subroutine CopyBlock is a utility routine used to copy the contents of a block of memory in the dynamic memory area. The input parameters for the subroutine are the pointer to the block to be copied and the size of the block. If the destination block does not already exist, then subroutine Gimme is used to create a new block. The output parameter is the pointer to the new copy of the block. This

routine copies over all contents of the block including pointers. Therefore, care must be taken when using this routine.

4.2.5 Subroutine LogMe

Subroutine LogMe logs the plan for the unit processing the message containing the list of plans. The input parameters are the pointer to the unit processing the message and the pointer to the current plan in the list of plans. The current plan in the list of plans is removed from the list and the pointer to the current plan is set to the next plan in the list. If the new plan for the unit is more recent than the current plan for the unit then the new plan replaces the old one; otherwise, the new plan is discarded. The output parameter is the pointer to the next plan in the list of plans.

4.2.6 Subroutine LogSub

Subroutine LogSub processes the plan for a subordinate unit of the unit processing a "plan" message. The input parameters are the pointer to the unit processing the message, the pointer to the unit's status block for the subordinate unit, the pointer to the output message process structure which contains the parameters for message

handling, the pointer to the output message format and the pointer to the current plan in the list of plans.

The current plan is logged within the unit's status block for the subordinate unit. After the plan is logged it is sent to the subordinate unit via a "plan" message. The message is created and put into the communications network by subroutine MsgOut. The pointer to the current plan is then set to the next plan in the list. The output parameter is the updated pointer to the current plan.

4.2.7 Subroutine MergeCmdr

Subroutine MergeCmdr merges the plans contained in "plan" messages sent by the processing unit's commander. The input parameters are the pointer to the top of the queue of "plan" messages, the identifier of the unit processing the messages, the pointer to the top of the current master list of plans, and the pointer to the bottom of the current master list of plans. It is assumed that any external messages in the queue of "plan" messages have already been processed by subroutine MergeExt. Each external message has an empty list as its list of plans. Therefore, all messages that have non-empty lists need to be processed.

Each plan contained within a message in this queue of "plan" messages must be processed. These plans are used to

create the secondary list of plans. Each plan is added to the secondary list unless one of two conditions holds: (1) the plan to be added is for a unit that already has a plan in the master list, or (2) the plan to be added is for a unit that already has a plan in the secondary list and the create time of the plan already there is more recent than the create time of the plan to be added. If a plan is not added to the secondary list then the appropriate message is sent to the user. If a plan is added to the secondary list due to the presence of another plan for the same unit but with an older create time, the plan already in the secondary list is removed. The output parameter is the pointer to the top of the secondary list.

4.2.8 Subroutine MergeExt

Subroutine MergeExt merges the plans contained within external "plan" messages into a master list of plans. The input parameters are the pointer to the queue of "plan" messages containing the external "plan" messages and the identifier of the node processing the messages. The queue of "plan" messages may contain "plan" messages that are not externally generated. These messages are not processed. If the queue does not have any external messages in it then the master list should be set to nil. Otherwise, the master

list should be set to the list of plans contained in the first external message found. The plans contained in any other external messages should be added to this master list, if appropriate. It is appropriate to add a plan to the master list if there is not already a plan in the master list for the same unit as the one being added. If it is not appropriate to add the plan to the master list, then subroutine MergeExt sends an error message to the user. The output parameters are the pointers to the top and bottom of the master list of plans. After the execution of this subroutine no external messages (in the queue of "plan" messages) are assumed to contain pointers to plans, i.e. the plans have been removed from the messages.

4.2.9 Subroutine MergePlan

Subroutine MergePlan merges the plans received at a node via "plan" messages into one master list of plans. The input parameter to this routine is the pointer to the node that has received the "plan" messages. The NODE data structure for this node contains the pointer to the root of the queue of messages to be processed. Four tasks are to be performed by subroutine MergePlan. The first task is to merge into a master list all of the plans received by external message. This task is performed by subroutine MergeExt. The second task is to merge into a secondary list

all plans received by non-external messages. This task is performed by subroutine MergeCmdr. The third task is to release the dynamic memory space used by the messages. The fourth task is to concatenate the secondary list onto the end of the master list. The latter two tasks are performed by subroutine MergePlan. The output parameter from this routine is the pointer to the top of the queue of the master list of plans.

4.2.10 Subroutine PrtLine

Subroutine PrtLine is used to print the plan for a single unit. The input parameter is the pointer to the plan block to be printed. Subroutine ChTime is used to convert a time value in model time to a character string containing the time in standard military format.

4.2.11 Subroutine PrtPlan

Subroutine PrtPlan prints out an updated plan for a particular node. The input parameters for this routine are the pointer to the node containing the updated plan to be printed and the pointer to the top of the queue of status blocks for this node. The plan for a node includes any available plans the unit has for its subordinate units. Each plan for a unit and its subordinate units is printed

individually on a single line. Subroutine PrtLine is used to print the plan for a single unit.

4.2.12 Subroutine ReadPlan

Subroutine ReadPlan is used to read in the additional data lines for an external "plan" message containing plans for red units. The input parameter is the pointer to the message that has the additional data lines. This subroutine is assumed to be undisturbed if no additional data lines exist for the message in question. The format of the line read in is (6X, I6, F7.3, 3I6, 2F7.3, I6).

The values read in are: the identifier of the plan's unit, the goal force ratio red/blue, the goal time, the periodic review time and delta, the low and high bounds for the force ratio red/blue, and the additional data flag. All but the additional data flag are stored in a plan block and added to the list of plans for the message. The additional data flag is used to determine if another additional data line should be added which is indicated by a flag value of one. A flag value of zero implies no additional data lines.

4.2.13 Subroutine XChngPlan

Subroutine XChngPlan is used by subroutine MergeCmdr to replace the contents of a plan block with the contents of another plan block. The input parameters are the pointers

to the source and destination plan blocks. Both the source and destination plan blocks are assumed to exist. All contents of the source block are copied over except the first location which contains the pointer to the next block. This location is not copied since the queue containing the destination block is not to be altered. Only the contents of the destination block are to be altered.

4.3 Function two: Combat unit reactions

The ability of a combat unit level commander (node of type 302) to evaluate his status with respect to his plan is achieved by rule number 3024. This rule is periodic with a period of one which allows the commander to evaluate his status every time step. Currently this is every 30 minutes. The activation of rule 3024 results in the calling of subroutine DivReact.

One existing DDS block has been modified and two blocks have been created. The block labeled CMBT has been modified so that it contains an element named IOPSTAT. This element is the index (into the combat operations status matrix) of the current combat status of the combat unit level node for which the CMBT block exists. The two new DDS blocks are labeled OPSTAT and PERMSG. The list of OPSTAT blocks makes up the combat operations status matrix which is global for

all units. This matrix contains pairs of posture and engagement rate indices. Each unit in combat has an index into this matrix and can change its posture and engagement rate values by changing its combat operations status index. The block labeled PERMSG contains a combat level unit commander's "reaction" message (message of type 3024) to his intermediate level commander. The message contains the old and new values for the combat unit's combat operations status index. The message also contains a flag indicating whether or not it is the unit's goal time.

The proper execution of this function requires initialization of the PLAN blocks for all units in combat. This is required because all units in combat must follow their plans, i.e., each must have a plan to follow. Currently, the plans are initialized by sending external "plan" messages at the beginning of combat. However, this method has its drawbacks. These drawbacks and a possible solution are discussed in Chapter 6.

The top level routine for this function is subroutine DivReact. First, subroutine DivReact calls subroutine INCombat to determine if the unit in question is in combat. If the unit is not in combat, then the unit's commander need not attempt to stay within any plan of action. If the unit is in combat, then DivReact determines if the unit has strayed from its plan. If the unit has strayed then

subroutine ChngPER is called to change the unit's combat operations status index, posture, and engagement rate values. Finally, if the unit needs to send a "reaction" message to its intermediate level commander then subroutine RptAction is called.

4.3.1 Subroutine DivReact

Subroutine DivReact allows a Red division that is in combat to check if it is staying within its plan. The input parameters for this routine are the pointer to the unit performing this process, the pointer to the output message process structure which contains the parameters for message handling, and the pointer to the output message format. A unit's plan consists of two parts. The first part is a goal time and a goal force ratio that is to be reached by the goal time. At a user specified delta before the goal time the unit checks if it has reached the goal force ratio. This delta is stored in COMMON's as global variable NReact. If the unit has not reached the goal, then it makes one last effort to reach the goal force ratio before the goal time. At goal time the unit sends a message to its commander so that the commander can evaluate its progress.

The second part of a unit's plan consists of staying within its prescribed force ratio bounds. If the unit's

force ratio crosses either the high or low bound, then the unit takes the appropriate corrective action and sends a message to its commander notifying him of the action taken. Corrective action consists of moving the unit's combat operations status index either up or down depending on whether the unit's force ratio is below the low bound or above the high bound, respectively. The changing of the combat operations status index also changes the values of the posture and engagement rate indices.

A unit is also required to send a periodic spot check report to its commander. This allows the commander to periodically evaluate the unit. However, the only time a periodic spot check report needs to be sent is when the unit takes no action. This is because any action taken results in the sending of a message to the commander, alerting him to the action taken.

4.3.2 Subroutine ChngPER

Subroutine ChngPER changes the posture and engagement rates for a particular node. This is done by changing the combat operations status index for the node and by retrieving the corresponding posture and engagement rate for the new combat operations status index. The first parameter for this routine is a flag indicating to choose the next larger or smaller combat operations status index. If the

current combat operations status index is at the end of the list, then the new values are the same as the old ones. The two other parameters are the pointer to the combat data structure for the unit in question and the identifier of the unit. The combat data structure is the storage location of the combat operations status index, the posture, and the engagement rate.

4.3.3 Subroutine RptAction

Subroutine RptAction is used by subroutine DivReact to send a division "reaction" message from a unit in combat to its commander. The input parameters are the pointer to the unit performing this process, the old and new combat operations status indices, the goal time check flag, the pointer to the output message process structure which contains the parameters for message handling, and the pointer to the output message format.

The message is sent by subroutine MsgOut. The contents of the additional data block for the message are: the old and new values for the unit's combat operations status index and the goal time check flag which indicates whether or not this is a "goal time" message. This message notifies the commander that he needs to evaluate the division's actions.

4.4 Function three: Intermediate commander reactions

The ability of an intermediate level commander to (1) evaluate a combat unit level commander's actions and (2) remove the combat commander (if appropriate) is achieved by rule number 4521. Rule 4521 allows an intermediate level commander to (1) receive "reaction" messages (messages of type 3024) from combat unit level commanders and (2) send two "change of commander" messages (messages of type 4521) to each combat unit requiring reprisals. The activation of this rule results in the calling of subroutine AGCReact.

The two "change of commander" messages simulate the replacement of the subordinate unit's commander. The first message is sent immediately and contains the subordinate unit's new combat operations status index. The second message is delayed according to the time delta specified in global variable NRecov which is stored in COMMONs. The second message also contains a combat operations status index. The first index represents the subordinate's posture and engagement rate while without a commander. The second index represents the subordinate's posture and the engagement rate to which it should return upon the arrival of the new commander.

Five DDS blocks are required for this function; four of these blocks already exist (or have been created to perform one of the two functions previously discussed). Block

PERMSG contains a "reaction" message from a subordinate unit (combat unit level commander). This message is used to evaluate the action taken by the combat unit level commander sending the message. Block NODE contains the pointer to the top of the queue of "reaction" messages to be processed by the intermediate level commander. The OPSTAT and STATUS blocks are used to evaluate the actions taken by the combat unit level commanders. The new DDS block CHNGCMDR is used to send a "change of commander" message to a combat unit.

The proper execution of this function requires initialization of the PLAN blocks contained within STATUS blocks since the intermediate level commanders need to verify that their subordinates are staying within their designated plans. In order to do this each intermediate level commander must have his own copy of each of the plans for each of his subordinate units. As mentioned in the previous section, the plans are currently initialized by sending external "plan" messages at the beginning of combat. However, this method has its drawbacks. These drawbacks and a possible solution are discussed in Chapter 6.

The top level routine for this function is subroutine AGCReact which allows an intermediate level commander (Army Group Commander) to verify the actions taken by subordinate units. Subroutine AGCReact repeatedly calls subroutine

AGCStaff to process each of the messages in the list of "reaction" messages received by the commander. Subroutine AGCStaff has one of two tasks to perform. Task One is performed if the message indicates the subordinate unit has reached its goal time. For Task One AGCStaff checks if the subordinate unit has reached its goal force ratio. If the unit has not reached its goal force ratio then subroutine AGCMsg is called to send two reprisal messages to the subordinate. If Task One is not performed then Task Two is performed; this means AGCStaff calls subroutine AGCAction to perform the same actions that the subordinate unit performed. If the results of the actions peformed by the subordinate are not the same as the results of the actions performed by AGCAction, then AGCStaff calls subroutine AGCMsg to send two reprisal messages to the subordinate.

4.4.1 Subroutine AGCReact

Subroutine AGCReact allows an Army Group Commander to verify the action taken by its subordinate units. The input parameters are the pointer to the unit performing this process, the pointer to the output message process structure which contains the parameters for message handling, and the pointer to the output message format. The list of division "reaction" messages is assumed to contain no more than one message from any one particular subordinate unit. This

means that subroutine MsgIn has already removed from the list any messages from a particular unit that has another message in the list with a more recent create time. Subroutine AGCStaff is called with each message in the list to verify the action taken by the subordinate unit. AGCStaff also performs any necessary reprisals.

4.4.2 Subroutine AGCAction

Subroutine AGCAction allows an Army Group Commander to perform the same actions that a subordinate unit performed. The input parameters for this routine are the pointer to the node performing this process, the pointer to the Army Group Commander's status block for the subordinate unit and the pointer to the Army Group Commander's plan block for the subordinate unit. The Army Group Commander performs these actions to verify that the subordinate unit has taken the appropriate actions.

The actions to be performed are: to check if the subordinate unit has crossed under its force ratio low bound, to check if the unit has crossed over its force ratio high bound, and to check if the unit should have sent a spot check report. The resultant effect of this routine is the modified value of the subordinate unit's combat operations status index. This value is stored within the status block

for the unit.

4.4.3 Subroutine AGCMsg

Subroutine AGCMsg sends two reprisal messages from an Army Group Commander to the appropriate subordinate unit. The input parameters are the pointer to the unit sending the two messages, the pointer to the output parameter message process structure which contains the parameters for message handling, the pointer to the output parameter message format and the pointer to the status block for the subordinate unit to which the messages are to be sent. Subroutine MsgOut is used to create the actual messages and to put them into the communications network.

4.4.4 Subroutine AGCStaff

Subroutine AGCStaff processes a single division "reaction" message received by an Army Group Commander (AGC). The input parameters for this routine are the pointer to the unit performing this process, the pointer to the output message process structure which contains the parameters for message handling, the pointer to the output message format and the pointer to the division "reaction" message to be processed. If the message received is a "goal time" message, then a check determines if the unit has reached its goal force ratio. If the unit has not reached

its goal force ratio, then AGC changes its combat operations status index and calls subroutine AGCMsg to send reprisal messages to that subordinate unit. If the message received is not a "goal time" message, then subroutine AGCAction is called to perform the same actions that the subordinate unit performed. If the results of the action are not the same as the results of the subordinate's action, then subroutine AGCMsg is called to send reprisal messages to the subordinate unit.

4.4.5 Subroutine ChngPER2

Subroutine ChngPER2 is similiar to subroutine ChngPER. This routine is used by subroutines AGCStaff and AGCAction to allow an Army Group Commander to perform the same actions its subordinate unit performed. The difference between this routine and ChngPER is the second parameter. Instead of using the combat data structure, this routine uses the Army Group Commander's status block for the subordinate unit in question.

4.5 Function four: Combat unit change of commander

The ability of a combat unit to change its commander is achieved by rule number 3026. This rule allows a combat unit to receive a "change of commander" message (message

type 4521) sent from the unit's intermediate level commander. This function requires three already existing DDS blocks (or blocks created to perform one of the functions previously discussed). The "change of commander" message being received is contained in block CHNGCMDR. Block OPSTAT is used to get the new posture and engagement rate values that correspond to the new combat operations status index indicated in the message. Block CMBT contains the combat operations status index, posture, and engagement rate values to be updated.

The top level routine for this function is subroutine ChngCmdr which is the only routine for this function (except for utility routines). The details of this function can be found in the section below on subroutine ChngCmdr.

4.5.1 Subroutine ChngCmdr

Subroutine ChngCmdr processes a reprisal (change of commander) message received by a unit. The input parameter is the pointer to the unit performing this process. The pointer to the message to be processed is located within the DDS block for the node performing the process. Only one "change of commander" message is assumed to exist for any one unit during a single time increment. This means that subroutine MsgIn takes the appropriate actions to resolve the reception of more than one message for a particular unit

during the same time increment. The message received contains the new combat operations status index for this unit. This new value is saved and is also used to determine and save the new posture and engagement rate values for this unit.

CHAPTER 5
Test Suite

5.1 Introduction

The following test suite is not all-inclusive for the C3Eval model. This research is a subcontract effort to the overall enhancement of the model to create the red commander functions. Therefore, sample results of the total system runs are not available. This test suite performs functional testing of each of the elements developed during the research. Each of the next four sections of this chapter describes the tests performed to verify one of the four functions developed. The final section is a sample run of the model containing tests for each of the four functions.

5.2 Logging plans

There are both "black box" and "white box" tests performed on this function. Both types of testing start with a base test case of the arrival and the processing of a "plan" message at the combat level or at the intermediate commander level. Both types of testing derive their test cases by modifying this base case.

At the black box level there are several cases to consider. The first case is the specification of the origin of the "plan" message being received. A "plan" message can

be generated either externally (external message) or internally (message from actual unit). A combat level unit can receive a message of either origin; however, an intermediate level commander is unable to receive an internally generated message, since the model currently does not have a high level commander to create the message. The second case is the number of plans contained within the "plan" message received. The message can contain: (1) one plan for the unit receiving the message, (2) one or more plans for subordinates of the unit receiving the message, or (3) one for the unit receiving the message and one or more plans for subordinates of the unit receiving the message. Note that combat level units do not have subordinates and therefore only receive plans for themselves.

At the white box level there are also several cases to consider. The first case is a plan that is not for the unit receiving the "plan" message and is not for a subordinate unit of the unit receiving the message, i.e., the reception of an "illegal" plan. The second case concerns a plan that is for a subordinate of the unit receiving the "plan" message. When this occurs the unit receiving the message should not only log the plan but should also send the plan to the appropriate subordinate unit. The third case concerns the reception of more than one "plan" message at a unit during one time increment. This case consists of: (1)

multiple plans for a unit that are externally generated, (2) multiple plans for a unit that are internally generated, and (3) one or more externally generated plans and one or more internally generated plans for a unit.

The list of test cases for this function is:

(1) Combat level unit receives externally generated "plan" message for self.
(2) Combat level unit receives internally generated "plan" message for self.
(3) Intermediate level unit receives externally generated "plan" message for self.
(4) Intermediate level unit receives externally generated "plan" message for subordinates and plans are sent to appropriate subordinates.
(5) Intermediate level unit receives externally generated "plan" message for self and subordinates and plans are sent to appropriate subordinates.
(6) A unit receives an "illegal" plan.
(7) Intermediate level unit receives multiple external "plan" messages during one time increment.
(8) Combat level unit receives externally generated "plan" message and internally generated "plan" message during one time increment.

5.3 Combat unit reactions

There are both "black box" and "white box" levels for viewing the tests performed on this function. At the black box level there are two types of messages generated by this function. The two types of messages are: (1) a message stating an action taken (one possible action is to take no action), and (2) a message stating that the unit sending the message has reached its goal time. The actual test cases for the first type of message are developed at the white box

level.

The white box level consists of three cases for the first type of message. The first type of message can be: (1) a periodic message stating that no action is taken, (2) a periodic message stating that an action is taken, or (3) a non-periodic message stating that an action is taken. A message stating that an action is taken indicates one of two situations has occurred, namely, (1) a combat level unit's force ratio has crossed under the unit's force ratio low bound, or (2) a combat level unit's force ratio has crossed over the unit's force ratio high bound.

The list of test cases for this function is:

(1) A periodic review message stating that no action is taken.
(2) A periodic review message stating that an action is taken. The message is also triggered by a unit crossing over its high bound.
(3) A non-periodic review message stating that an action is taken. The message is triggered by a unit crossing under its low bound.
(4) A message stating that a unit has reached its goal time.

5.4 Intermediate commander reactions

There are both "black box" and "white box" levels for viewing the tests performed on this function. At the black box level there are two types of messages that are received by this function. The two types of messages are: (1) a message stating an action taken by a subordinate (one

possible action is to take no action), and (2) a message stating that the subordinate unit has reached its goal time. The actual test cases for the first type of message are developed at the white box level. There are two test cases for the second type of message. The first case is that the intermediate commander determines that the subordinate unit has reached its goal force ratio. In this case no further action is taken. The second case is that the intermediate commander determines that the subordinate unit has not reached its goal force ratio. In this case the intermediate commander sends two reprisal messages to the subordinate unit in question.

The white box level for this function consists of two cases: (1) the intermediate level commander agrees with the action taken, and (2) the intermediate level commander disagrees with the action taken. The action taken can be one of three actions: (1) no action, (2) changing combat operations status index because unit's force ratio crossed under force ratio low bound, or (3) changing combat operations status index because unit's force ratio crossed over force ratio high bound.

The list of test cases for this function is:

(1) Intermediate commander receives message and agrees with the action taken.
(2) Intermediate commander receives message and disagrees with the action taken. Commander sends two reprisal

messages to the appropriate subordinate unit.
(3) Intermediate commander receives message and determines unit has reached its goal force ratio.
(4) Intermediate commander receives message and determines unit has not reached its goal force ratio. Commander sends two reprisal messages to the appropriate subordinate unit.
(5) Intermediate commander receives more than one message during one time increment and properly processes each message independently.

5.5 Combat unit change of commander

For this function the black box and white box approaches produce the same test cases. There are only two test cases for this function. The first case is a combat level unit receiving a "change of commander" message that indicates the removal of the unit's commander. The second case is a combat level unit receiving a "change of commander" message that indicates the arrival of the unit's new commander. Both cases produce the same result which is the changing of the unit's combat operations status index.

The list of test cases for this function is:

(1) Combat level unit receives a "change of commander" message indicating the removal of its commander.
(2) Combat level unit receives a "change of commander" message indicating the arrival of its new commander.

5.6 Example run

This section describes the test cases performed in the example run. The example run contains test cases for each of the four functions listed above. For the example run the

intermediate level commander is unit number 12 which is labeled ARMY GP TAC. The combat level unit is unit number 1 which is labeled 155 MRD. The output has been truncated to only include information pertinent to the test cases described (Figure 6).

For the remainder of this section the extracted outputs are from Figure 6. In the discussion that follows, test cases from above included in the example run are identified in the following manner: test case "section number.test case number". For example, test case 5.1 is the first test case in the list of test cases for the combat unit change of commander function (discussed as Section 5 above).

```
EXTERNAL MSG AT TIME   13 MESSAGE TYPE 6540 FROM UNIT TYPE  652 TO UNIT ARMY GP TAC CREATED AT TIME   13
        MESSAGE DATA   12 23.456   20    2    4  1.500  3.499 1
        MESSAGE DATA   20  3.400   35    3    5  1.000  3.600 1
        MESSAGE DATA    1  2.800   19   32   24  2.825  3.150 0
EXTERNAL MSG AT TIME   13 MESSAGE TYPE 4520 FROM UNIT TYPE  452 TO UNIT 155 MRD     CREATED AT TIME   13
        MESSAGE DATA    1  2.800   19   32   24  2.825  3.150 0

TIME   13 INPUT   QUEUE ARMY GP TAC TYPE 6540 FROM INPUT    PRIORITY  1 CREATED AT   13 SEND AT   13
TIME   13 INPUT   QUEUE 155 MRD     TYPE 4520 FROM INPUT    PRIORITY  1 CREATED AT   13 SEND AT   13

LOGPLAN FINDS NO STATUS BLOCK FOR NODE   12 SUBORDINATE   20

PLAN UPDATE AT DAY 00 TIME 0630
PLAN FOR ARMY GP TAC

    UNIT        |GOAL FORCE/RATIO|        TIME        |        START        |PERIODIC REVIEW DELTA|  FORCE RATIO LOW  |  HIGH  |
-------------------------------------------------------------------------------------------------------------------------------
ARMY GP TAC          23.456     DAY 00 TIME 1000    DAY 00 TIME 0100    DAY 00 TIME 0200        1.500          3.499
155 MRD               2.800     DAY 00 TIME 0930    DAY 00 TIME 1600    DAY 00 TIME 1200        2.825          3.150

PLAN UPDATE AT DAY 00 TIME 0630
PLAN FOR   155 MRD

    UNIT        |GOAL FORCE/RATIO|        TIME        |        START        |PERIODIC REVIEW DELTA|  FORCE RATIO LOW  |  HIGH  |
-------------------------------------------------------------------------------------------------------------------------------
155 MRD               2.800     DAY 00 TIME 0930    DAY 00 TIME 1600    DAY 00 TIME 1200        2.825          3.150

TIME   13 FUTURE  QUEUE ARMY GP TAC  TYPE 4520 TO 155 MRD     PRIORITY  1 CREATED AT   13 SEND AT   16

TIME   14 FUTURE  QUEUE ARMY GP TAC  TYPE 4520 TO 155 MRD     PRIORITY  1 CREATED AT   13 SEND AT   16

TIME   15 FUTURE  QUEUE ARMY GP TAC  TYPE 4520 TO 155 MRD     PRIORITY  1 CREATED AT   13 SEND AT   16

UNIT 201 ACR        COMBAT STRENGTHS
                ENG RATE - BLUE 3/1   RED 3/3   TIME   15   R/B FORCE RATIO  2.819

TIME   16 OUTPUT  QUEUE ARMY GP TAC  ON LINK  1 MESSAGE 4520 TO 155 MRD      PRIORITY  1 CREATED   13 SEND   16
TIME   16 OUTPUT  QUEUE 155 MRD      ON LINK  1 MESSAGE 3024 TO ARMY GP TAC  PRIORITY  2 CREATED   16 SEND   16
```

Figure 6
Truncated output for example run

```
                    COMBAT STRENGTHS
UNIT 201 ACR        ENG RATE - BLUE 3/1   RED 3/3   TIME   16   R/B FORCE RATIO   2.835

TIME   17 INPUT   QUEUE ARMY GP TAC  TYPE 3024 FROM 155 MRD      PRIORITY  2 CREATED AT   16 SEND AT   16
TIME   17 INPUT   QUEUE 155 MRD      TYPE 4520 FROM ARMY GP TAC  PRIORITY  1 CREATED AT   13 SEND AT   16

PLAN UPDATE AT DAY 00 TIME 0830
PLAN FOR 155 MRD

    UNIT        |GOAL FORCE/RATIO|        TIME        |        PERIODIC REVIEW START        |        DELTA        | FORCE RATIO LOW | HIGH |
   155 MRD              2.800     DAY 00 TIME 0930     DAY 00 TIME 1600     DAY 00 TIME 1200        2.825       3.150

TIME   17 OUTPUT  QUEUE ARMY GP TAC  ON LINK  1 MESSAGE 4521 TO 155 MRD      PRIORITY  1 CREATED   17 SEND   17
TIME   17 FUTURE  QUEUE ARMY GP TAC  TYPE 4521 TO 155 MRD      PRIORITY  1 CREATED AT   17 SEND AT 18
TIME   18 INPUT   QUEUE 155 MRD      TYPE 4521 FROM ARMY GP TAC  PRIORITY  1 CREATED AT   17 SEND AT   17
REMOVAL OF COMMANDER FOR NODE #    1. NEW COMBAT OPERATIONS STATUS INDEX IS     2
TIME   18 OUTPUT  QUEUE ARMY GP TAC  ON LINK  1 MESSAGE 4521 TO 155 MRD      PRIORITY  1 CREATED   17 SEND   18
TIME   19 INPUT   QUEUE 155 MRD      TYPE 4521 FROM ARMY GP TAC  PRIORITY  1 CREATED AT   17 SEND AT   18
ARRIVAL OF COMMANDER FOR NODE #    1. NEW COMBAT OPERATIONS STATUS INDEX IS     2
TIME   19 OUTPUT  QUEUE 155 MRD      ON LINK  1 MESSAGE 3024 TO ARMY GP TAC  PRIORITY  2 CREATED   19 SEND   19

                    COMBAT STRENGTHS
UNIT 201 ACR        ENG RATE - BLUE 3/1   RED 3/3   TIME   19   R/B FORCE RATIO   2.873
TIME   20 INPUT   QUEUE ARMY GP TAC  TYPE 3024 FROM 155 MRD      PRIORITY  2 CREATED AT   19 SEND AT   19
```

Figure 6 (continued)
Truncated output for example run

The example run contains four test cases for the logging of plans. The four test cases are: 2.1, 2.2, 2.5, and 2.6. Test case 2.1 is represented in the output by the external message sent to 155 MRD at time 13. The extracted output is:

```
EXTERNAL MSG AT TIME   13 MESSAGE TYPE 4520 FROM UNIT TYPE  452 TO UNIT 155 MRD     CREATED AT TIME   13
        MESSAGE DATA    1  2.800   19   32   24  2.825  3.150 0

TIME   13 INPUT   QUEUE 155 MRD      TYPE 4520 FROM INPUT    PRIORITY  1 CREATED AT   13 SEND AT   13

PLAN UPDATE AT DAY 00 TIME 0630
PLAN FOR   155 MRD
```

UNIT	GOAL FORCE/RATIO	TIME	PERIODIC REVIEW START	DELTA	FORCE RATIO LOW	HIGH
155 MRD	2.800	DAY 00 TIME 0930	DAY 00 TIME 1600	DAY 00 TIME 1200	2.825	3.150

Test cases 2.5 and 2.6 are both initiated by one external message sent to ARMY GP TAC at time 13. Note that unit number 20 is not a subordinate of ARMY GP TAC. The extracted output is:

```
EXTERNAL MSG AT TIME   13 MESSAGE TYPE 6540 FROM UNIT TYPE  652 TO UNIT ARMY GP TAC CREATED AT TIME   13
        MESSAGE DATA   12 23.456   20    2    4  1.500  3.499 1
        MESSAGE DATA   20  3.400   35    3    5  1.000  3.600 1
        MESSAGE DATA    1  2.800   19   32   24  2.825  3.150 0

TIME   13 INPUT   QUEUE ARMY GP TAC TYPE 6540 FROM INPUT    PRIORITY  1 CREATED AT   13 SEND AT   13

LOGPLAN FINDS NO STATUS BLOCK FOR NODE   12 SUBORDINATE   20

PLAN UPDATE AT DAY 00 TIME 0630
PLAN FOR ARMY GP TAC
```

UNIT	GOAL FORCE/RATIO	TIME	PERIODIC REVIEW START	DELTA	FORCE RATIO LOW	HIGH
ARMY GP TAC	23.456	DAY 00 TIME 1000	DAY 00 TIME 0100	DAY 00 TIME 0200	1.500	3.499
155 MRD	2.800	DAY 00 TIME 0930	DAY 00 TIME 1600	DAY 00 TIME 1200	2.825	3.150

```
TIME   13 FUTURE  QUEUE ARMY GP TAC  TYPE 4520 TO 155 MRD     PRIORITY  1 CREATED AT   13 SEND AT   16
TIME   14 FUTURE  QUEUE ARMY GP TAC  TYPE 4520 TO 155 MRD     PRIORITY  1 CREATED AT   13 SEND AT   16
TIME   15 FUTURE  QUEUE ARMY GP TAC  TYPE 4520 TO 155 MRD     PRIORITY  1 CREATED AT   13 SEND AT   16
TIME   16 OUTPUT  QUEUE ARMY GP TAC  ON LINK  1 MESSAGE 4520 TO 155 MRD     PRIORITY  1 CREATED   13 SEND   16
```

Test case 2.2 is represented by 155 MRD receiving at time 13 the "plan" message sent by ARMY GP TAC. The extracted output is:

```
TIME   17 INPUT   QUEUE 155 MRD      TYPE 4520 FROM ARMY GP TAC  PRIORITY  1 CREATED AT   13 SEND AT   16

PLAN UPDATE AT DAY 00 TIME 0830
PLAN FOR 155 MRD
```

UNIT	GOAL FORCE/RATIO	TIME	PERIODIC REVIEW START	DELTA	FORCE RATIO LOW	HIGH
155 MRD	2.800	DAY 00 TIME 0930	DAY 00 TIME 1600	DAY 00 TIME 1200	2.825	3.150

The example run contains two test cases for combat level unit reactions. Test case 3.3 is seen in the output when the 155 MRD realizes at time 16 that it has crossed under its low bound and sends a message of type 3024 to its intermediate level commander (ARMY GP TAC). The force ratio for the 155 MRD is seen in the output as the red/blue (r/b) force ratio for the 201 ACR since the 201 ACR is the 155 MRD's opponent. The extracted output is:

```
                COMBAT STRENGTHS
UNIT 201 ACR    ENG RATE - BLUE 3/1   RED 3/3   TIME   15   R/B FORCE RATIO  2.819
TIME   16 OUTPUT  QUEUE 155 MRD     ON LINK  1 MESSAGE 3024 TO ARMY GP TAC PRIORITY  2 CREATED   16 SEND   16
```

Test case 3.4 is seen in the output when at time 19 the 155 MRD sends a message of type 3024 to ARMY GP TAC. This message is signaling ARMY GP TAC that 155 MRD has reached its goal time. The extracted output is:

```
TIME   19 OUTPUT  QUEUE 155 MRD     ON LINK  1 MESSAGE 3024 TO ARMY GP TAC  PRIORITY  2 CREATED   19 SEND   19
```

The example run contains two test cases for intermediate level commander reactions. Test case 4.2 is seen in the output when ARMY GP TAC receives at time 17 a message of type 3024 sent by 155 MRD. Then ARMY GP TAC sends two reprisal messages (type 4521) to 155 MRD at times 17 and 18. These reprisal messages are sent because the intermediate level commander looks at the 155 MRD's force ratio one time unit later than when the unit looks and determines that the unit is above its low bound. The extracted output is:

```
                    COMBAT STRENGTHS
UNIT 201 ACR      ENG RATE - BLUE 3/1   RED 3/3   TIME   16   R/B FORCE RATIO   2.835
TIME   17 INPUT   QUEUE ARMY GP TAC  TYPE 3024 FROM 155 MRD      PRIORITY  2 CREATED AT   16 SEND AT   16
TIME   17 OUTPUT  QUEUE ARMY GP TAC  ON LINK  1 MESSAGE 4521 TO 155 MRD      PRIORITY  1 CREATED   17 SEND   17
TIME   17 FUTURE  QUEUE ARMY GP TAC  TYPE 4521 TO 155 MRD      PRIORITY  1 CREATED AT   17 SEND AT 18
TIME   18 OUTPUT  QUEUE ARMY GP TAC  ON LINK  1 MESSAGE 4521 TO 155 MRD      PRIORITY  1 CREATED   17 SEND   18
```

Test case 4.3 is seen in the output when ARMY GP TAC receives at time 20 a message of type 3024 sent by 155 MRD. This message signals that 155 MRD has reached its goal time. The lack of any reprisal messages sent by ARMY GP TAC signifies that ARMY GP TAC has determined that 155 MRD has reached its goal force ratio. The extracted output is:

```
                    COMBAT STRENGTHS
UNIT 201 ACR      ENG RATE - BLUE 3/1   RED 3/3   TIME   19   R/B FORCE RATIO   2.873
TIME   20 INPUT   QUEUE ARMY GP TAC  TYPE 3024 FROM 155 MRD      PRIORITY  2 CREATED AT   19 SEND AT   19
```

The example run contains both test cases for the combat unit change of commander. Test case 5.1 is seen in the output at time 18 when 155 MRD (unit number 1) receives a message of type 4521 and removes its current commander. The extracted output is:

```
TIME   18 INPUT   QUEUE 155 MRD      TYPE 4521 FROM ARMY GP TAC  PRIORITY  1 CREATED AT   17 SEND AT   17
REMOVAL OF COMMANDER FOR NODE #    1. NEW COMBAT OPERATIONS STATUS INDEX IS     2
```

Test case 5.2 is seen in the output at time 19 when 155 MRD receives another message of type 4521 which is the arrival of the new commander. The extracted output is:

```
TIME   19 INPUT   QUEUE 155 MRD      TYPE 4521 FROM ARMY GP TAC  PRIORITY  1 CREATED AT   17 SEND AT   18
ARRIVAL OF COMMANDER FOR NODE #    1. NEW COMBAT OPERATIONS STATUS INDEX IS     2
```

CHAPTER 6
Summary and Critique

6.1 Critique of the existing model

During the course of the research, several problems with the model have been identified that justify further attention. This section gives a brief description of these problems.

One of the difficulties with the programming technique used in the C3Eval model concerns the use of the dynamically allocated virtual memory space. The allocation and deallocation of memory requires knowledge of the size of the block in question. The references to values stored in the memory require knowledge of the offset within the block of the value in question. This knowledge is hardcoded into the model. This situation can be improved by creating an electronic dictionary containing the description of all the DDS blocks. This dictionary has two interfaces with the FORTRAN code. The first interface is a function which takes a block name as the input parameter and returns the size of the block. The second interface is a function which takes a block name and an element name as input parameters and returns either the index or offset of the element of the block in question. The difference between an index and an offset is one. For example, an index of one has an offset

value of zero.

Another problem with the C3Eval model is that units (nodes) do not have the ability to anticipate messages. A unit should be able to expect a message, and if the message does not arrive, then the unit can take the appropriate action. For example, a red combat level unit should acknowledge the receipt of a "plan" message. However, if the intermediate level commander is not able to expect the acknowledgement, then there is no purpose in sending it.

A third problem with the C3Eval model is that each rule must have an output format if the rule requires specialized processing to be performed by a subroutine. This is because the routines called within subroutine Process to perform the specialized processing are identified by their output format. For example, rule number 3020 (combat level unit logging a "plan" message) must have an output format even though the rule never creates output messages.

6.2 Critique of the project

Two elements of the research project deserve further attention. The first element concerns initialization. Like all simulation models and expert systems, initialization is very important and is often the difference between a valid and invalid system. The proper execution of the C3Eval model (with the new enhancements from the research) requires

that each combat level unit (in combat) has a plan. The intermediate level commander of each of these units is required to have his own copy of the unit's plan. This requirement forces each unit in combat to attempt to stay within its plan, i.e., no real combat unit goes into combat without some sort of plan of action. Therefore, initialization must be performed for at least all the units that are in combat at the beginning of the simulation. Currently, this is performed by sending external messages to all combat level units (in combat at the beginning of the simulation) and to the intermediate level commander of each of these units. The problem is that the intermediate commander then sends the message (four time steps later) to the subordinate unit. This is unnecessary as well as inappropriate for initialization. Therefore, a data set should be created that initializes all appropriate plans.

The second element of the research project that requires attention is the algorithm used when a combat level unit does not reach its goal. Currently, each unit at goal time minus combat reaction time checks to see if it has met its goal force ratio. If the unit has not met its goal then the unit changes its posture and engagement rate by changing its index within the combat operations status matrix. In addition to this, the unit should commit any reserve forces

that it has. These reserve forces represent the portion of the combat unit that is not in combat, i.e., the portion of the unit that is not currently engaging the enemy.

6.3 Future enhancements

The enhancements discussed below pertain only to the red side of the model. The previous two sections discussed the need for modifications/extensions to existing elements of the model. This section discusses the need for new elements to be added to the model.

The next extension to the red side of the C3Eval model beyond this research project is the commitment of reserve forces by the intermediate level commander. These reserve forces are in the form of second echelon combat units (divisions currently not in combat). The contents of the plan for an intermediate level commander are expanded to accomodate the new tasks he must perform. This also requires the creation of an actual node for a high level commander. This new node is responsible for the performance of the intermediate level commanders. Such a responsiblity includes sending intermediate level commanders their plans as well as verifying that the commanders are performing adequately. This is similiar to the interaction between the combat unit level commanders and the intermediate level commanders.

Another enhancement is to give each unit its own copy of the combat operations status matrix. This allows different units to react in different manners to changing status; these changes reflect the differing personalities of the commanders. Once this is implemented the "plan" messages are modified so that a unit's combat operations status matrix can be changed. This allows for the representation of either (1) the replacement of the commander for a particular node, or (2) the changing attitude of the commander for the particular node. Such a representation introduces another scenario where a subordinate and his commander disagree on the appropriate action to be taken.

Three suggested enhancements are similar to the previous one. The first enhancement is to implement a user specified combat operations stability time. This means a unit does not check its status with respect to its plan for the specified amount of time after taking corrective measures for straying from the plan. This keeps a unit from overreacting. The second enhancement is to allow a unit to move more than one step within a combat operations status matrix. Currently, if a unit has strayed from its plan, then it moves up or down one location within the combat operations status matrix. A modification to base the

permissible move on the distance from the unit's plan appears to be more realistic.

The third enhancement combines the previous two enhancements to allow stability time to vary depending on the size of the step taken within the combat operations status matrix, i.e. the larger the move taken within the matrix, the longer a unit waits before checking his status again. This allows the unit to delay for observation of the results of the action already taken.

The following two suggested enhancements modify the plan of action followed by the combat units. One enhancement adds another goal to those that are to be reached by the designated goal time. The new goal to be added is for each combat level unit to be at a designated index within the combat operations status matrix. The new goal allows for the representation of several similar concepts, one of which is that a unit must be in the appropriate posture and engagement rate at its goal time so that the unit can initiate action toward the enemy. The second enhancement is to use trend analysis on the unit's force ratio in addition to control limits. Currently, each combat level unit checks its force ratio with respect to its designated low and high bounds. This is improved by adding the use of moving averages and smoothing to detect trends which indicate that a unit's force ratio may cross over one

of the two bounds.

Finally, the red side needs to be added to the preprocessor and postprocessor. For the preprocessor this includes the nodes, communications links, external messages, and any other elements of the red side that have been duplicated from the blue side. This also includes any additional data sets created for the red side, for example, the data set for the initialization of PLAN blocks. For the postprocessor, all capabilities to create graphs for the blue side are duplicated for the red side.

BIBLIOGRAPHY

Anderson, Lowell Bruce, "An Initial Postulation of a Relatively General Attrition Process," IDA working paper WP-20, Project 2371, 1984.

Ashby, W. Ross. An Introduction to Cybernetics. London: Methuen & Co. Ltd., 1979.

Charniak, Eugene and Drew McDermott. Introduction to Artificial Intelligence. Addison-Wesley Publishing Company, 1986.

Chiu, Yu-Hsing, et al., "The Simulation of Command and Control in the Conflict Model (CONMOD)," Preprint UCRL 97281, Lawrence Livermore National Laboratory, 1987.

Connell, John, Lester Ingber and Charles Yost, "A Statistical Mechanical Virtual Neural Computer and C3 Applications," Proceedings: 1987 Joint Directors of Laboratories C2 Research Symposium, pp.49-57.

Darcom Pamphlet, Engineering Design Handbook: Army Weapon Systems Analysis, Part Two, (Department of the Army, Headquarters U.S. Army Materiel Development and Readiness Command, Alexandria, VA, 1979), pp. 28-1 to 28-51.

George, F. H. The Foundations of Cybernetics. New York: Gordon and Breach Science Publishers, 1977.

Herres, General Robert, "Remarks on the Importance of Command and Control Research," Proceedings: 1987 Joint Directors of Laboratories C2 Research Symposium,pp.3-5.

Ingber, Lester, "C3 Decision Aids: Statistical Mechanics Application of Biological Intelligence," Proceedings: 1987 Joint Directors of Laboratories C2 Research Symposium, pp.49-57.

Ingber, Lester and Stephen Upton, "A Stochastic Model of Combat," Proceedings: 1987 Joint Directors of Laboratories C2 Research Symposium, pp.49-57.

Klir, Jiri and Miroslav Valach. Cybernetic Modelling. London: ILIFFE Books Ltd., 1967.

Robinson, Robert, Joseph Stahl and Merle Roberson, "Command and Control Effectiveness through Combat Outcomes," Signal, July 1987, pp.75-85.

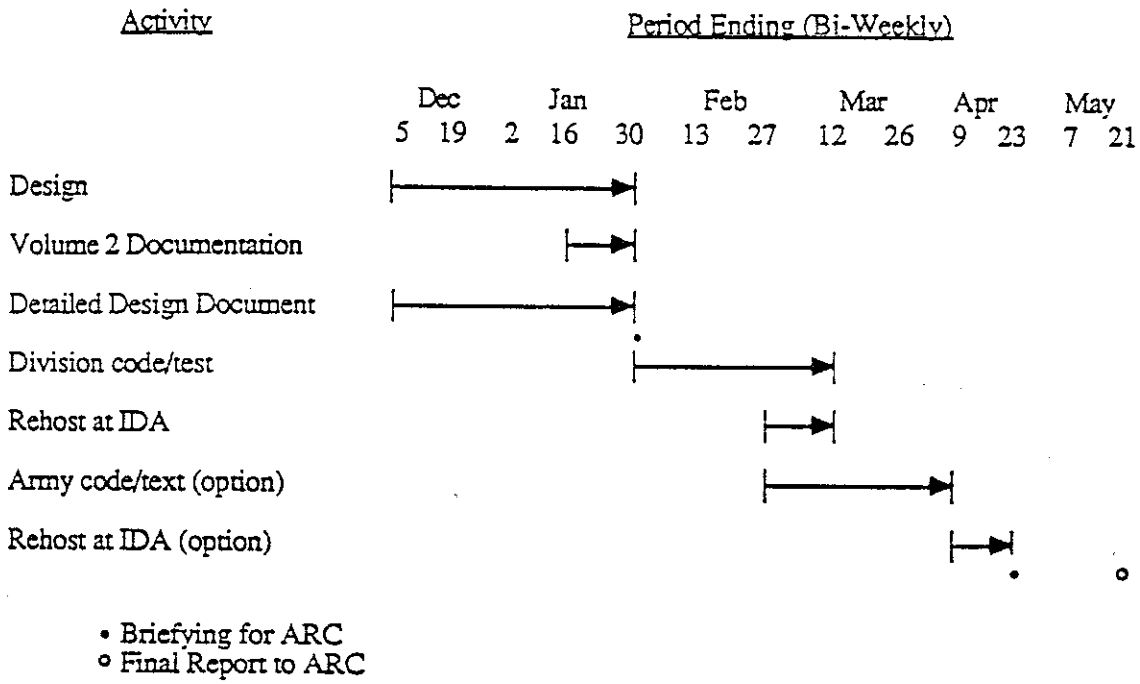
Rudall, B. H. Computers and Cybernetics. Kent: Abacus Press, 1981.

Trappl, Robert. Cybernetics: Theory and Applications. Hemisphere Publishing Corporation, 1983.

APPENDIX A
Glossary

ARC	Applications Research Corporation
CAA	Combined Arms Army
CAS	Close air support
CINC	Commander in Chief
C2	Command and control
C3	Command, control and communications
DBMS	Data base management system
DDS	Dynamic data structures
F/R	Force ratio (for the purposes of this paper it is assumed the force ratio is red/blue)
IDA	Institute for Defense Analyses
JCS	Joint Chiefs of Staff

APPENDIX B
Schedule and Milestones



• Briefying for ARC
o Final Report to ARC

APPENDIX C
Dynamic Data Structures

The dynamic data structures (DDS's) are the implementation of the linked list data blocks that are required to provide essentially dimensionless code. Each logical set of DDS's has its root in a non-dynamic location and is linked together by pointers. These structures are created during input by calls to Subroutine GIMME (already existing code) which acquires data blocks from dynamic memory. The length of each type data block is a fixed number in the code. This section lists the new data blocks, defines their elements, gives the type of each variable, identifies the routine(s) that creates the block and the ones that delete the block if applicable and specifies the location of the root. The symbols used in the DDS's documented in this section are:

Symbol	Type of Element
P	Pointer to another DDS
I	Integer value
R	Real value
C	Character

The majority of the new DDS's are linked with the DDS labeled NODER which is the NODE structure for the red side. The other new DDS has its own root which is located in the new COMMON's called ENGMT2. The links for the new DDS's are shown in Figures 7 and 8.

COMMON/C3/NODER

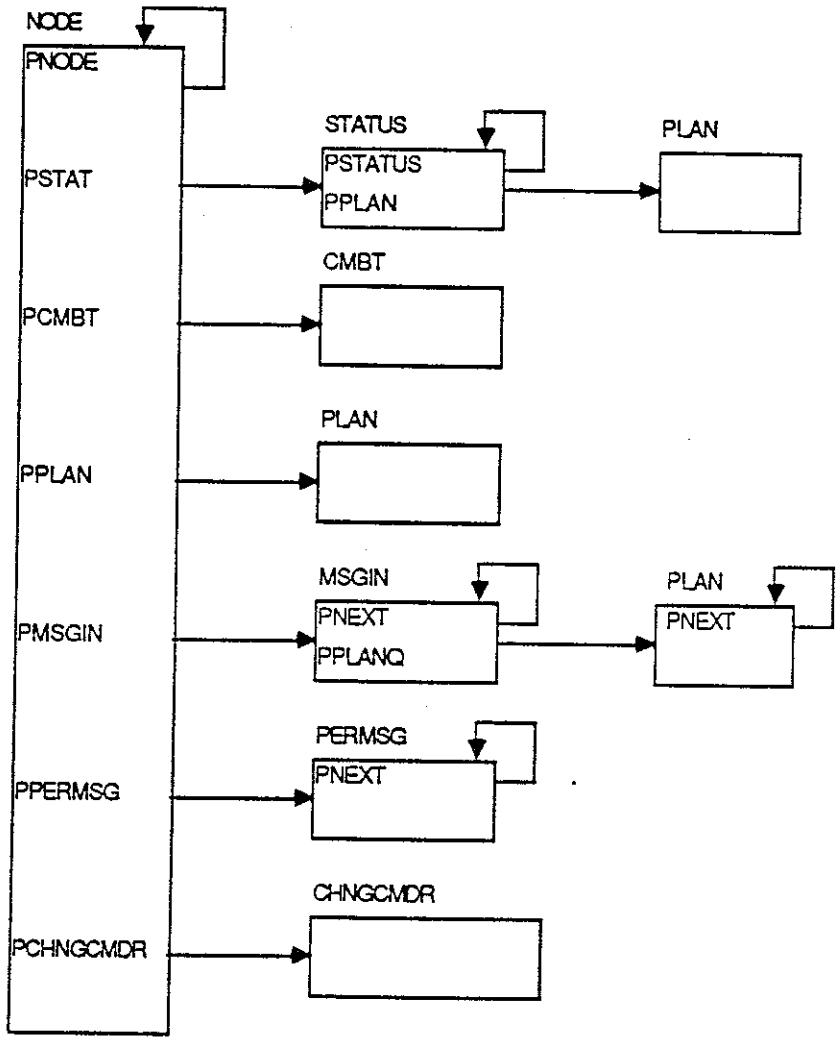


Figure 7 Modifications to Node Dynamic Linking

COMMON/C3/ENGMT2

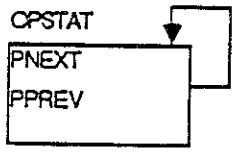


Figure 8 Combat operations status matrix

BLOCK NAME: ChngCmdr BLOCK SIZE: 21

USE: Message sent as replacement orders for a subordnate's commander due to not following the plan.

CREATED BY: AGCMSG
DELETED BY: AGCMSG, CHNGCMDR
ROOT: NODE(30) DATE: 29 Feb 88

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PNEXT	P	Pointer to next ChngCmdr
2	ORIG	I	Originator unit identifier
3	CTIME	I	Create time
4	SUBORDID	I	Subordinate unit's identifier
5	IOPSTAT	I	New combat operations status index
6	IFLAG	I	Purpose of message: =0 remove commander; =1 assignment of commander
8-21			Not used

BLOCK NAME: CMBT BLOCK SIZE: 20

USE: The combat data structure contains the combat weapon systems data for each combat level unit

CREATED BY: INPUTC
DELETED BY: N/A
ROOT: NODE(9) DATE: 29 Feb 88

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	CFLAG	I	Not used
2	ENGB	I	Engagement rate of Blue
3	ENGR	I	Engagement rate of Red
4	FRB	R	Force ratio Red/Blue
5	POSB	I	Posture of Blue
6	POSR	I	Posture of Red
7	PSYSB	P	Pointer to Blue systems
8	PSYSR	P	Pointer to Red systems
9	XB	I	X location of Blue
10	YB	I	Y location of Blue
11	XR	I	X location of Red
12	YR	I	Y location of Red
13	PSUPPLY	P	Pointer to arriving supply queue
14	TIMEA	I	Emergency AMMO request time
15	TIMEP	I	Emergency POL request time

BLOCK NAME: CMBT (continued) BLOCK SIZE:

USE:

CREATED BY:
DELETED BY:
ROOT: DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
16	TIMES	I	Emergency SUPPLY request time
17	IOPSTAT	I	Current Red combat operations status
18-20			Not used

BLOCK NAME: MSGIN BLOCK SIZE: 7

USE: Message summary data containing a list of new plans to be logged

CREATED BY: MSGIN
DELETED BY: MERGEPLAN
ROOT: NODE(28)

DATE: 29 Feb 88

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PNEXT	P	Pointer to next message summary
2	MSGTYPE	I	Message type
3	ORIG	I	Originator unit identifier
4	CTIME	I	Creation time
5	PPLAN	P	Pointer to list of additional data
6-7			Not used

BLOCK NAME: NODE

BLOCK SIZE: 30

USE: This structure is the top element for each node. It is used to locate the functional structures that contain the node's characteristics and data.

CREATED BY: INPUT

DELETED BY: Not applicable

ROOT: COMMON/C3/NODE1

DATE: 29 Feb 88

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PNODE	P	Pointer to next node
2	ID	I	Unit identifier number
3	TYPE	I	Unit type
4	PDEST	P	Pointer to destination queue
5	POMP	P	Pointer to output message process
6	PIMQ	P	Pointer to input message queue
7	PFMQ	P	Pointer to future message queue
8	PCMDR	P	Pointer to commander
9	PCMBTC	P	Pointer to combat data structure
10	PAOPS	P	Pointer to assigned air support
11	PREQ	P	Pointer to requests for support
12	PCOUNT	P	Pointer to counters for output
13-15	NAME	C	Unit name
16	PATR	P	Pointer to acknowledged requests
17	PFSO	P	Pointer to requested fire support

BLOCK NAME: NODE (continued)

BLOCK SIZE:

USE:

CREATED BY:
DELETED BY:
ROOT:

DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
18	PFSR	P	Pointer to approved fire support
19	PSTAT	P	Pointer to subordinate status queue
20	PSPOT	P	Pointer to spot report queue
21	PALLOC	P	Pointer to allocation parameters
22	PFRQ	P	Pointer to force ratio queue
23	PRMQ	P	Pointer to random message queue
24	PRND	P	Pointer to random distributions
25	PSUP	P	Pointer to support data structure
26	PPLAN	P	Pointer to Red unit's plan
27			Not used
28	PMSGIN	P	Pointer to list of plan messages (Red side only)
29	PPERMSG	P	Pointer to list division action messages (Red side only)
30	PCHGNCMDR	P	Pointer to change of commander message (Red side only)

BLOCK NAME: OPSTAT BLOCK SIZE: 10

USE: Contains the operational status of combat level units

CREATED BY: REDCDS
DELETED BY: N/A
ROOT: COMMON/ENGMT2/POPSTATQ DATE: 29 Feb 88

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PNEXT	P	Pointer to next operational status
2	PPREV	P	Pointer to previous operational status
3	IOPSTAT	I	Combat operations status index
4	POSTURE	I	Posture index
5	ENGRATE	I	Engagement rate index
6	LOSTAT	I	IOPSTAT value during commander replacement
7	NXSTAT	I	IOPSTAT value after commander replacement
8-10			Not used

BLOCK NAME: PERMSG BLOCK SIZE: 21

USE: Subordinate's report of action taken

CREATED BY: RPTACTION
DELETED BY: RPTACTION
ROOT: NODE(29) DATE: 29 Feb 88

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PNEXT	P	Pointer to next PERMSG
2	UNITID	I	Unit identifier of subordinate that changed status values
3	TIME	I	Time that change was made
4	IOPSTATO	I	Old operations status index value
5	IOPSTATN	I	New operations status index value
6	GCHECK	I	Goal time check flag, =0 No; =1 Yes
7-21			Not used

BLOCK NAME: PLAN

BLOCK SIZE: 21

USE: Contains a plan of action for a unit or it's subordinate

CREATED BY: READPLAN, LOGSUB
DELETED BY: ADDEXT, LOGME, LOGSUM, MERGECMDR
ROOT: STATUS(54), NODE(26)

DATE: 29 Feb 88

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PNEXT	P	Pointer to next PLAN
2	ORIG	I	Originator of plan
3	CTIME	I	Time plan was created
4	SUBORDID	I	Subordinate unit's identification
5	FRBGOAL	R	Goal force ratio value
6	GTIME	I	Time to reach goal
7	PRTIME	I	Periodic review time
8	PRDELTA	I	Periodic review delta
9	FRBLOW	R	Force ratio low bound
10	FRBHIGH	R	Force ratio upper bound
11-21			Not used

BLOCK NAME: STATUS

BLOCK SIZE: 55

USE: Status of subordinate units, created if subordinate flag in input = 2.

CREATED BY: INPUT
DELETED BY: N/A
ROOT: NODE(19)

DATE: 29 Feb 88

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PSTAT	P	Pointer to next status structure
2	UNITID	I	Identifier of status unit
3	TIMEB	I	Last time before status updated
4-14	NB	I	Number of Blue weapons
15-25	BLOSS	R	Number of Blue losses
26	TIMER	I	Last time Red status updated
27	PFOE	P	Pointer to Red foe structure
28-38	RLOSS	R	Number of Red losses
39	ENGB	I	Blue engagement rate
40	ENGR	I	Red engagement rate
41	FRB	R	Red to Blue force ratio
42	PFRQ	P	Pointer to next status structure in force ratio queue
43	LFRT	I	Last time force ratio calculated
44	PCASRPT	P	Pointer to CAS report
45	CASIMED	I	Immediate CAS flag

BLOCK NAME: STATUS (continued)

BLOCK SIZE:

USE:

CREATED BY:
DELETED BY:
ROOT:

DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
46	USEA	I	Current AMMO resupply rate
47	USEP	I	Current POL resupply rate
48	USER	I	Current PARTS resupply rate
49	TIMEA	I	Next time for AMMO train schedule
50	TIMES	I	Next time for SUPPLY train schedule
51	ERFA	I	Emergency AMMO request flag
52	ERFP	R	Emergency POL multiplier
53	ERFS	I	Emergency PARTS request flag
54	PPLAN	P	Pointer to subordinate unit's plan (Red side only)
55	IOPSTAT	I	Current combat operations status (Red side only)

APPENDIX D
Source Code

```
      SUBROUTINE AddEnd(PPlan, Top, Bottom)
C
C     PURPOSE:  Add a plan to the end of a list of plans.
C
C     PARAMETERS:
C     (INPUT)
C        PPlan    ==> Pointer to plan to add to list.
C     (INPUT/OUTPUT)
C        Top      ==> Pointer to top of list.
C        Bottom   ==> Pointer to bottom of list.
C
      IMPLICIT INTEGER (P)
      INTEGER Top, Bottom
      INCLUDE 'Include.for'

      IF (Bottom.EQ.0) THEN
C
C        New plan is the only entry in the list.
C
         Top = PPlan
         Memory(PPlan) = 0
         Bottom = PPlan
      ELSE
C
C        Add new plan to the end of the list.
C
         Memory(Bottom) = PPlan
         Memory(PPlan) = 0
         Bottom = PPlan
      END IF
      RETURN
      END
```

```
      SUBROUTINE AddExt(NodeId, PMsgIn, MasterT, MasterB)
C
CC    PURPOSE:  Add the plans contained within an external
CC    message to the master list of plans.  If a plan is for
CC    a unit that already has a plan in the master list then
CC    do not add new plan and send error message to user.
CC
CC    PARAMETERS:
CC    (INPUT)
CC       NodeId  ==> Identifier of node processing messages.
CC       PMsgIn  ==> Pointer to external message containing
CC                      plans to add to master list.
CC    (INPUT/OUTPUT)
CC       MasterT ==> Top of queue of merged plans.
CC       MasterB ==> Bottom of queue of merged plans.
C
      IMPLICIT INTEGER (P)
      INCLUDE 'Include.for'

      PPlan = Memory(PMsgIn + 4)
      Memory(PMsgIn + 4) = 0
C
CC    Do for each plan contained in the external message.
C
1400  CONTINUE
      IF (PPlan.EQ.0) GO TO 1500
         PNext = Memory(PPlan)
         CALL Find(MasterT, 3, Memory(PPlan+3), PTemp)
         IF (PTemp.NE.0) THEN
C
CC          Plan already exists in master list.  Send error
CC          message.
C
            WRITE(IOut, 1420) NodeId, Memory(PPlan+3)
1420        FORMAT(1X,'* ERROR * AddExt - node ',I4,
     -           ' received multiple plans for node ',I4)
            CALL Releas(PPlan, 21, Memory)
         ELSE
C
CC          Plan does not already exist in master list. Add
CC          plan to end of master list.
C
            CALL AddEnd(PPlan, MasterT, MasterB)
         END IF
         PPlan = PNext
      GO TO 1400
1500  CONTINUE
      RETURN
      END
```

```
      SUBROUTINE AGCAction(PNode, PStatus, PPlan)
C
CC    PURPOSE:  Allow Army Group Commander to perform same
CC    actions that subordinate unit performed. The commander
CC    is performing these actions to verify the action taken
CC    by the subordinate.
CC
CC    PARAMETERS:
CC    (INPUT)
CC       PNode   ==> Pointer to node performing this process
CC       PStatus ==> Pointer to status block for subordinate
CC       PPlan   ==> Pointer to plan block for subordinate
C
      IMPLICIT INTEGER (P)
      INCLUDE 'Include.for'
C
CC    If unit has crossed under force ratio low bound then
CC    increment combat operations status index and save new
CC    posture and engagement rate indices.
C
      IF (Store(PStatus+40).LT.Store(PPlan+8)) THEN
         CALL ChngPER2(0, PStatus, Memory(PNode+1))
C
CC    If unit has crossed over force ratio high bound then
CC    decrement combat operations status index and save new
CC    posture and engagement rate indices.
C
      ELSE IF (Store(PStatus+40).GT.Store(PPlan+9)) THEN
         CALL ChngPER2(1, PStatus, Memory(PNode+1))
C
CC    Checks if unit should have sent a spot check report.
C
      ELSE IF (ITime.GE.Memory(PPlan+6)) THEN
         Memory(PPlan+6) = Memory(PPlan+6) + Memory(PPlan+7)
      END IF
      RETURN
      END
```

```
      SUBROUTINE AGCMsg(PNode, POMP, POut, PStatus)
C
CC    PURPOSE:  Send two reprisal messages from the Army
CC    Group Commander to the appropriate subordinate unit.
CC    These two messages simulate the replacement of the
CC    subordinate unit's commander.
CC
CC    PARAMETERS:
CC    (INPUT)
CC       PNode   ==> Pointer to unit doing this process.
CC       POMP    ==> Pointer to the process.
CC       POut    ==> Pointer to the output message format.
CC       PStatus ==> Pointer to status structure for
CC                   subordinate.
C
      IMPLICIT INTEGER (P)
      INCLUDE 'Include.for'
C
CC    Find the pointer to the entry in the matrix for the
CC    current value of the combat operations status index
CC    for the subordinate unit.
C
      CALL Find(POpStatQ, 2, Memory(PStatus+54), POpStat)
C
CC    Send a reprisal message to the subordinate unit.  The
CC    message contains the new value for the units combat
CC    operations status index.  This represents the removal
CC    of the division commander.
C
      CALL Gimme(PChngCmdr, 21, Memory)
      Memory(PChngCmdr+1) = Memory(PNode+1)
      Memory(PChngCmdr+2) = ITime
      Memory(PChngCmdr+3) = Memory(PStatus+1)
      Memory(PChngCmdr+4) = Memory(POpStat+5)
      Memory(PChngCmdr+5) = 0
      CALL MsgOut(PNode, POMP, POut, PChngCmdr, 21, 1)
C
CC    Send another reprisal message to the subordinate unit.
CC    This message is delayed according to user specified
CC    time delta. The message contains the combat operations
CC    status index to return to. This represents the arrival
CC    of the new division commander.
C
      Memory(PChngCmdr+2) = ITime + NRecov
      Memory(PChngCmdr+4) = Memory(POpStat+6)
      Memory(PChngCmdr+5) = 1
      Memory(POMP+6) = Memory(POMP+6) + NRecov
      CALL MsgOut(PNode, POMP, POut, PChngCmdr, 21, 1)
C
```

```
C       Reset the time to process the message. Return the
C       memory used to free space.
C
        Memory(POMP+6) = Memory(POMP+6) - NRecov
        CALL Releas(PChngCmdr, 21, Memory)
        RETURN
        END
```

```
      SUBROUTINE AGCReact(PNode, POMP, POut)
C
CC    PURPOSE:  Allow Army Group Commander to verify action
CC    taken by subordinate units. This subroutine assumes
CC    that the list of division action messages received for
CC    the current unit contains no duplicates.  This means
CC    there should not be more than one division reaction
CC    message from a particular subordinate.
CC
CC    PARAMETERS:
CC    (INPUT)
CC       PNode    ==> Pointer to unit doing this process.
CC       POMP     ==> Pointer to the process.
CC       POut     ==> Pointer to the output message format.
C
      IMPLICIT INTEGER (P)
      INCLUDE 'Include.for'

      PPERMsg = Memory(PNode+28)
      Memory(PNode+28) = 0
C
CC    Do for each message in list.
C
100   CONTINUE
      IF (PPERMsg.EQ.0) GO TO 1000
         PNext = Memory(PPERMsg)
C
CC       Verify action taken by subordinate. If action taken
CC       was incorrect then send two reprisal messages. The
CC       first one removes the division commander and the
CC       second one replaces the division commander.

         CALL AGCStaff(PNode, POMP, POut, PPERMsg)
         PPERMsg = PNext
         GO TO 100
1000  CONTINUE
      RETURN
      END
```

```
      SUBROUTINE AGCStaff(PNode, POMP, POut, PPERMsg)
C
C     PURPOSE:  Process division reaction message received
C     by Army Group Commander that was sent by subordinate
C     unit. If subordinate has failed to stay within the
C     plan then take appropriate actions.
C
C     PARAMETERS:
C     (INPUT)
C        PNode   ==> Pointer to unit doing this process.
C        POMP    ==> Pointer to the process.
C        POut    ==> Pointer to the output message format.
C        PPERMsg ==> Pointer to input message to process.

      IMPLICIT INTEGER (P)
      INCLUDE 'Include.for'
      CALL Find(Memory(PNode+18), 1, Memory(PPERMsg+1),
     -          PStatus)
      IF (PStatus.NE.0) THEN
         PPlan = Memory(PStatus + 53)
         IF (Memory(PPERMsg+5).EQ.0) THEN
C
C           Perform same actions that subordinate took. If
C           results of actions are not the same as action
C           taken by subordinate then replace the units
C           commander by sending two reprisal messages.

            CALL AGCAction(PNode, PStatus, PPlan)
            IF (Memory(PStatus+54).NE.Memory(PPERMsg+4))THEN
               CALL AGCMsg(PNode, POMP, POut, PStatus)
            END IF
         ELSE IF (Store(PStatus+40).LT.Store(PPlan+4)) THEN
C
C           If unit should be at goal force ratio and is not
C           then increment combat operations status index &
C           save new posture and engagement rate indices.
C           Also replace the subordinates commander by
C           sending two reprisal messages to subordinate.

            CALL ChngPER2(0, PStatus, Memory(PNode+1))
            CALL AGCMsg(PNode, POMP, POut, PStatus)
         END IF
      ELSE
         WRITE(IOut, *) '* ERROR * AGCREACT - UNIT # ',
     -        Memory(PNode+1),' HAS NO SUBORDINATE # ',
     -        Memory(PPERMsg+1)
      END IF
      RETURN
      END
```

```
      SUBROUTINE ChngCmdr(PNode)
C
CC    PURPOSE:  Process change of commander message received
CC    by a unit. Message contains the new combat operations
CC    status index for this unit. It is assumed there will
CC    be only one change of commander message to process for
CC    any one unit.
CC
CC    PARAMETERS:
CC    (INPUT)
CC       PNode    ==> Pointer to unit processing message.
CC
      IMPLICIT INTEGER (P)
      INCLUDE 'Include.for'
C
CC    Get change of commander message.
C
      PChngCmdr = Memory(PNode+29)
      Memory(PNode+29) = 0
      IF (PChngCmdr.NE.0) THEN
C
CC       Save new combat operations status index, posture
CC       and engagement rate.
C
         IOpStat = Memory(PChngCmdr+4)
         PCmbt = Memory(PNode+8)
         Memory(PCmbt+16) = IOpStat
         CALL Find(POpStatQ, 2, IOpStat, POpStat)
         IF (POpStat.NE.0) THEN
            Memory(PCmbt+5) = Memory(POpStat+3)
            Memory(PCmbt+2) = Memory(POpStat+4)
         END IF
C
CC       Write out message stating combat operations status
CC       index has been changed due to change of commander.
C
         IF (Memory(PChngCmdr+5).EQ.0) THEN
            WRITE(IOut,100)'REMOVAL',Memory(PNode+1),IOpStat
         ELSE
            WRITE(IOut,100)'ARRIVAL',Memory(PNode+1),IOpStat
         END IF
100      FORMAT(1X,A7,' OF COMMANDER FOR NODE # ',I4,'. NEW'
     -      ,' COMBAT OPERATIONS STATUS INDEX IS ',I4)
         CALL Releas(PChngCmdr, 21, Memory)
      END IF
      RETURN
      END
```

```
      SUBROUTINE ChngPER(IFlag, PCmbt, NodeId)
C
C     PURPOSE:  Change posture and engagement rate to new
C     values. New values are found by finding current combat
C     operations status index in list and getting next or
C     previous index depending on whether force ratio was
C     low or high. Also save new combat operations status
C     index.
C
C     PARAMETERS:
C     (INPUT)
C        IFlag    ==> Flag indicating whether to get next or
C                     previous index. 0 ==> get next index
C                     1 ==> get previous index.
C        PCmbt    ==> Pointer to combat data structure.
C        NodeId   ==> Identifier of unit performing process.
C
      IMPLICIT INTEGER (P)
      INCLUDE 'Include.for'
C
C     Find location of current combat operations status.
C
      CALL Find(POpStatQ, 2, Memory(PCmbt+16), POpStat)

      IF (POpStat.EQ.0) THEN
C
C        If current index not found in list then send error
C        message.
C
         WRITE(IOut, 100) Memory(PCmbt+16), NodeId
100      FORMAT(1X,'* ERROR * ChngPER - Combat operations ',
     -        'status index #',I4,' not found for unit # ',I4)
      ELSE IF (Memory(POpStat+IFlag).NE.0) THEN
C
C        If current index found and its not the end of the
C        list then save new index, posture and engagement
C        rate values.
C
         POpStat = Memory(POpStat+IFlag)
         Memory(PCmbt+16) = Memory(POpStat+2)
         Memory(PCmbt+5) = Memory(POpStat+3)
         Memory(PCmbt+2) = Memory(POpStat+4)
      END IF
      RETURN
      END
```

```
      SUBROUTINE ChngPER2(IFlag, PStatus, NodeId)
C
C     PURPOSE:  Change posture and engagement rate to new
C     values. New values are found by finding current combat
C     operations status index in list and getting next or
C     previous index depending on whether force ratio was
C     low or high. Save new combat operations status index.
C
C     PARAMETERS:
C     (INPUT)
C        IFlag   ==> Flag indicating whether to get next or
C                    previous index. 0 ==> get next index
C                    1 ==> get previous index.
C        PStatus ==> Pointer to status structure for
C                    subordinate.
C        NodeId  ==> Identifier of unit performing process.
C
      IMPLICIT INTEGER (P)
      INCLUDE 'Include.for'
C
C     Find location of current combat operations status.
C
      CALL Find(POpStatQ, 2, Memory(PStatus+54), POpStat)

      IF (POpStat.EQ.0) THEN
C
C        If current index not found in list then send error
C        message.
C
         WRITE(IOut, 100) Memory(PStatus+54), NodeId
100      FORMAT(1X,'* ERROR * CHNGPER2 - COMBAT OPERATIONS',
     -       'STATUS INDEX #',I4,' NOT FOUND FOR UNIT # ',I4)
      ELSE IF (Memory(POpStat+IFlag).NE.0) THEN
C
C        If current index found and its not the end of the
C        list then save new index, posture and engagement
C        rate values.
C
         POpStat = Memory(POpStat+IFlag)
         Memory(PStatus+54) = Memory(POpStat+2)
         Memory(PStatus+39) = Memory(POpStat+4)
      END IF
      RETURN
      END
```

```
      SUBROUTINE CopyBlock (POld, Size, PNew)
C
C     PURPOSE:  Copy the contents of a block. WARNING: This
C     subroutine copies over all contents of the block. This
C     includes pointers.  Therefore, care must be taken when
C     using this routine.
C
C     PARAMETERS:
C     (INPUT)
C        POld     ==> Pointer to block to be copied.
C        Size     ==> Size of block.
C     (OUTPUT)
C        PNew     ==> Pointer to new copy of block.
C
      IMPLICIT INTEGER (P)
      INTEGER Size
      INCLUDE 'Include.For'
C
C     If new block does not exist then create new block.
C
      IF (PNew.EQ.0) THEN
         CALL Gimme(PNew, Size, Memory)
      END IF
C
C     For each element of the block, copy the virtual memory
C     that contains the integers and the virtual memory that
C     contains the reals.
C
      DO 100 Pos = 0, Size-1, 1
         Memory(PNew + Pos) = Memory(POld + Pos)
         Store(PNew + Pos) = Store(POld + Pos)
100   CONTINUE
      RETURN
      END
```

```
      SUBROUTINE DivReact(PNode, POMP, POut)
C
C     PURPOSE:  Red division checks if it is staying within
C     the plan as specified by plan block. If not staying
C     within the plan then take appropriate action and send
C     message to commander.
C
C     PARAMETERS:
C     (INPUT)
C        PNode    ==> Pointer to unit doing this process.
C        POMP     ==> Pointer to the process.
C        POut     ==> Pointer to the output message format.

      IMPLICIT INTEGER (P)
      INTEGER GCheck
      INCLUDE 'Include.for'
C
C     Check if unit is in combat. A value of one for NCombat
C     means the unit is in combat. A unit not in combat does
C     not attempt to stay within a plan.

      CALL INCombat(PNode, NCombat)
      IF (NCombat.EQ.1) THEN
         PPlan = Memory(PNode + 25)
         PCmbt = Memory(PNode + 8)
         IOpStatO = Memory(PCmbt+16)
         ISend = 1
         GCheck = 0
C
C        If it is goal time minus combat reaction time and
C        unit has not reached goal force ratio then make
C        last ditch effort to reach goal. This is done by
C        incrementing combat operations status index and
C        saving new posture and engagement rate indices.
C        Do not send message to commander. Need to write
C        subroutine to commit reserves.

         IF ((ITime.EQ.(Memory(PPlan+5)-NReact)).AND.
     -       (Store(PCmbt+3).LT.Store(PPlan+4))) THEN
               CALL ChngPER(0, PCmbt, Memory(PNode+1))
               ISend = 0
C
C        If it is goal time then send message to commander
C        so that he can evaluate units progress.

         ELSE IF (ITime.EQ.Memory(PPlan+5)) THEN
            GCheck = 1

C
```

```
C
C       If unit has crossed under force ratio low bound
C       then increment combat operations status index and
C       save new posture and engagement rate indices.

        ELSE IF (Store(PCmbt+3).LT.Store(PPlan+8)) THEN
           CALL ChngPER(0, PCmbt, Memory(PNode+1))

C
C       If unit has crossed over force ratio high bound
C       then decrement combat operations status index and
C       save new posture and engagement rate indices.

        ELSE IF (Store(PCmbt+3).GT.Store(PPlan+9)) THEN
           CALL ChngPER(1, PCmbt, Memory(PNode+1))
C
C       Checks if unit should send a spot check report.

        ELSE IF (ITime.GE.Memory(PPlan+6)) THEN
           Memory(PPlan+6)=Memory(PPlan+6)+Memory(PPlan+7)
        ELSE
           ISend = 0
        END IF
C
C       If appropriate, send message to commander.

        IF (ISend.EQ.1) THEN
           CALL RptAction(PNode, IOpStatO,Memory(PCmbt+16),
     -                    GCheck, POMP, POut)
        END IF
      END IF
      RETURN
      END
```

```
      SUBROUTINE LogMe (PNode, PPlan)
C
C     PURPOSE:  Log the plan for unit receiving the message.
C
C     PARAMETERS:
C     (INPUT)
C        PNode    ==> Pointer to unit to log plan for.
C     (INPUT/OUTPUT)
C        PPlan    ==> Pointer to current plan in the list of
C                     plans contained within the message.
C
      IMPLICIT INTEGER (P)
      INCLUDE 'Include.for'

      IF (Memory(PNode+25).NE.0) THEN
         IF (Memory(PPlan+2).GT.Memory(Memory(PNode+25)+2))
     -      THEN
C
C           Node has a plan that is older than new plan.
C           Remove new plan from list and save as plan for
C           the node. Make the next plan in the list the
C           current plan.
C
            CALL Releas(Memory(PNode+25), 21, Memory)
            Memory(PNode+25) = PPlan
            PPlan = Memory(PPlan)
            Memory(Memory(PNode+25)) = 0
         ELSE
C
C           Node has plan that is more recent than new plan.
C           Remove new plan from list. Make the next plan in
C           the list the current plan.
C
            PTemp = Memory(PPlan)
            CALL Releas(PPlan, 21, Memory)
            PPlan = PTemp
         END IF
      ELSE
C
C        Node doesn't have a plan. Remove new plan from list
C        and save as plan for the node. Make the next plan
C        in the list the current plan.
C
         Memory(PNode+25) = PPlan
         PPlan = Memory(PPlan)
         Memory(Memory(PNode+25)) = 0
      END IF
      RETURN
      END
```

```
      SUBROUTINE LogPlan (PNode, POMP, POut)
C
C     PURPOSE:  Log a plan for a Red division or a Red Army
C     Group Commander. Message may contain a list of plans;
C     one for the unit and one for each of its subordinates.
C
C     PARAMETERS:
C     (INPUT)
C        PNode    ==> Pointer to unit doing this process.
C        POMP     ==> Pointer to the process.
C        POut     ==> Pointer to the output message format.
C
      IMPLICIT INTEGER (P)
      INCLUDE 'Include.For'
C
C     Merge plan lists from messages being processed into 1
C     master list. Set current plan to first plan in list.
C
      CALL MergePlan(PNode, PPlan)
      IF (PPlan.EQ.0) GOTO 600
C
C     Do for each plan in the list. Determine if plan is for
C     unit receiving the message or one of its subordinate
C     units. Log plan in appropriate location and if plan is
C     for subordinate unit then send message containing the
C     plan to the subordinate. If plan is not for current
C     unit or its subordinates then send warning message and
C     continue on.
C
400   CONTINUE
      IF (PPlan.EQ.0) GO TO 500
         CALL Find(Memory(PNode+18), 1, Memory(PPlan+3),
     -              PStatus)
         IF (PStatus.EQ.0) THEN
            IF (Memory(PPlan+3).EQ.Memory(PNode+1)) THEN
               CALL LogMe(PNode, PPlan)
            ELSE
               WRITE(IOut, 430) Memory(PNode+1),
     -                          Memory(PPlan+3)
430            FORMAT(1X,'LogPlan finds no status block ',
     -              'for node ', I4,'    subordinate ',I4)
               PPlan = Memory(PPlan)
            END IF
         ELSE
            CALL LogSub(PNode, PStatus, POMP, POut, PPlan)
         END IF
      GO TO 400
500   CONTINUE
C
```

```
C       Print out the updated plan for the current node.
C
        CALL PrtPlan(PNode, Memory(PNode+18))
600     CONTINUE
        RETURN
        END
```

```
      SUBROUTINE LogSub(PNode, PStatus, POMP, POut, PPlan)
C
C     PURPOSE:  Log plan for subordinate of unit receiving
C     message. Send plan in a message to appropriate unit.
C
C     PARAMETERS:
C     (INPUT)
C        PNode   ==> Pointer to unit doing this process.
C        PStatus ==> Pointer to status block of subordinate.
C        POMP    ==> Pointer to the process.
C        POut    ==> Pointer to the output message format.
C     (INPUT/OUTPUT)
C        PPlan   ==> Pointer to the current plan in the list
C                    of plans contained within the message.

      IMPLICIT INTEGER (P)
      INCLUDE 'Include.for'
C
C     Save pointer to next plan in the list.
C
      PNext = Memory(PPlan)
C
C     If new plan is more recent than existing plan then
C     replace plan with new one and send message to unit.
C
      IF ((Memory(PStatus+53).EQ.0).OR.
     -   (Memory(PPlan+2).GT.Memory(Memory(PStatus+53)+2)))
     -   THEN
C
C        Set next pointer for current plan to nil. Log plan.
C
         Memory(PPlan) = 0
         CALL CopyBlock(PPlan, 21, Memory(PStatus+53))
C
C        Create new message containing current plan and send
C        to appropriate subordinate.
C
         Memory(PPlan+1) = Memory(PNode+1)
         Memory(PPlan+2) = ITime + Memory(POMP+6) - 1
         CALL MsgOut(PNode, POMP, POut, PPlan, 21, 1)
      END IF
C
C     Return the plan block to free space. Set current plan
C     to next plan in list of plans.
C
      CALL Releas(PPlan, 21, Memory)
      PPlan = PNext
      RETURN
      END
```

```
      SUBROUTINE MergeCmdr(PMsgQ, NodeId, MasterT, MasterB,
     -   PSecondT)
C
C     PURPOSE:  Merge commander messages into one list.  If
C     plan for a particular unit exists in external message
C     list then do not add to commander list. If more than
C     one plan for a particular unit exists then choose one
C     with the most recent time value.
C
C     PARAMETERS:
C     (INPUT)
C        PMsgQ     ==> Pointer to top of queue of messages
C        NodeID    ==> Identifier of node processing messages
C        MasterT   ==> Top of queue of external message plans
C        MasterB   ==> Bottom of queue of external message
C                      plans.
C     (OUTPUT)
C        PSecondT ==> Top of queue of merged commander plans
C
      IMPLICIT INTEGER (P)
      INCLUDE 'Include.for'

      PSecondT = 0
      PSecondB = 0
      PMsgIn = PMsgQ
C
C     Do for each message in list.
C
100   CONTINUE
      IF (PMsgIn.EQ.0) GO TO 5000
         PPlan = Memory(PMsgIn + 4)
C
C        If no list of plans then skip over this message. It
C        is assumed that external messages have already been
C        merged and their list of plans removed. If list of
C        plans then do for each plan block in list.
C
130      CONTINUE
         IF (PPlan.EQ.0) GO TO 4500
            PNext = Memory(PPlan)
            CALL Find(MasterT, 3, Memory(PPlan+3), PTemp)
            IF (PTemp.NE.0) THEN
C
C              Plan is for a unit that already has a plan in
C              the external message list. Don't save second
C              plan and send warning message to user.
C
               WRITE(IOut, 140) NodeId, Memory(PPlan+3)
```

```
 140            FORMAT(1X,'* WARNING * MergeCmdr - node ',I4,
     -              ' received multiple plans for node ',I4)
                CALL Releas(PPlan, 21, Memory)
             ELSE
                CALL Find(PSecondT,3,Memory(PPlan+3),PTemp)
                IF (PTemp.NE.0) THEN
C
C                  Plan is for unit that already has plan in
C                  the secondary list. Keep the plan with the
C                  most recent time value and send message.
C
                   WRITE(IOut, 150) NodeId, Memory(PPlan+3)
 150               FORMAT(1X,'* NOTE * MergeCmdr - node ',I4,
     -                ' received multiple plans for node',I4)
                   IF(Memory(PPlan+2).GT.Memory(PTemp+2))THEN
                      CALL XChngPlan(PPlan, PTemp)
                   END IF
                   CALL Releas(PPlan, 21, Memory)
                ELSE
C
C                  Have not found another plan for this unit.
C                  Add plan to secondary list of plans.
C
                   CALL AddEnd(PPlan, PSecondT, PSecondB)
                END IF
             END IF
             PPlan = PNext
          GO TO 130
 4500     CONTINUE
          PMsgIn = Memory(PMsgIn)
       GO TO 100
 5000  CONTINUE
       RETURN
       END
```

```
      SUBROUTINE MergeExt(PMsgQ, NodeId, MasterT, MasterB)
C
C     PURPOSE:  Merge external plan messages for a node into
C     one list. If more than one plan for a particular unit
C     exists then choose one nondeterministically.
C
C     PARAMETERS:
C     (INPUT)
C        PMsgQ   ==> Pointer to queue of plan messages.
C        NodeId  ==> Identifier of node processing messages.
C     (OUTPUT)
C        MasterT ==> Top of queue of merged plans.
C        MasterB ==> Bottom of queue of merged plans.
C
      IMPLICIT INTEGER (P)
      INCLUDE 'Include.for'
C
C     Find first external message in list.  Save the list of
C     plans for the message as the current master list.
C
      CALL Find(PMsgQ, 2, 99, PMsgIn)
      IF (PMsgIn.EQ.0) THEN
         MasterT = 0
         MasterB = 0
      ELSE
         MasterT = Memory(PMsgIn + 4)
         PTemp = MasterT
500      CONTINUE
            MasterB = PTemp
            PTemp = Memory(PTemp)
         IF (PTemp.NE.0) GO TO 500
         Memory(PMsgIn + 4) = 0
         END IF
C
C     For each additional external message add list of plans
C     contained in message to the master list of plans. If a
C     plan being added is already in list then don't add it.
C
1450  CONTINUE
      IF (PMsgIn.EQ.0) GO TO 1500
         CALL Find(Memory(PMsgIn), 2, 99, PNext)
         IF (PNext.NE.0) THEN
            CALL AddExt(NodeId, PNext, MasterT, MasterB)
         END IF
         PMsgIn = PNext
      GO TO 1450
1500  CONTINUE
      RETURN
      END
```

```
      SUBROUTINE MergePlan(PNode, MasterT)
C
C     PURPOSE:  If a node has more than one plan message to
C     process during one time increment then merge the plan
C     messages into one list.
C
C     PARAMETERS:
C     (INPUT)
C        PNode       ==> Pointer to node to merge plans for.
C     (OUTPUT)
C        MasterT     ==> Top of master queue of plans.

      IMPLICIT INTEGER (P)
      INCLUDE 'Include.for'

      NodeId = Memory(PNode + 1)
      PMsgQ = Memory(PNode + 27)
C
C     Merge external messages into one list.
C
      CALL MergeExt(PMsgQ, NodeId, MasterT, MasterB)
C
C     Merge commander messages into one list.
C
      CALL MergeCmdr(PMsgQ,NodeId,MasterT,MasterB,PSecondT)
C
C     Return the memory space used by messages to free space
C
      Memory(PNode + 27) = 0
100   CONTINUE
      IF (PMsgQ.EQ.0) GO TO 400
         PMsgIn = PMsgQ
         PMsgQ = Memory(PMsgIn)
         CALL Releas(PMsgIn, 7, Memory)
      GO TO 100
400   CONTINUE
C
C     Concatenate external message list and the commander
C     lists together.
C
      IF (MasterB.NE.0) THEN
         Memory(MasterB) = PSecondT
      ELSE
         MasterT = PSecondT
      END IF
      RETURN
      END
```

```
      SUBROUTINE PrtLine (PPlan)
C
C     PURPOSE:  Print the plan for one unit.
C
C     PARAMETERS:
C     (INPUT)
C        PPlan    ==> Pointer to plan block to be printed.
C
      IMPLICIT INTEGER (P)
      INCLUDE 'Include.for'
      CHARACTER*17 Time1, Time2, Time3

      IF (PPlan.EQ.0) GO TO 1000
C
C        Use subroutine ChTime to convert time increments
C        into day/time format.
C
         CALL ChTime(Memory(PPlan+5), Time1)
         CALL ChTime(Memory(PPlan+6), Time2)
         CALL ChTime(Memory(PPlan+7), Time3)
C
C        Use subroutine Find to get name of node plan is for
C
         CALL Find(Node1, 1, Memory(PPlan+3), PNode)
         WRITE(IOut, 500) (Memory(PNode+Pos), Pos = 12, 14),
     -      Store(PPlan+4), Time1, Time2, Time3,
     -      Store(PPlan+8), Store(PPlan+9)
500      FORMAT(2X, 3A4, 4X, F7.3, 4X, A17, 3X, A17, 3X,
     -      A17, 3X, F7.3, 3X, F7.3)
1000  CONTINUE
      RETURN
      END
```

```
      SUBROUTINE PrtPlan (PNode, PStatQ)
C
C     PURPOSE: Print out updated plan for a particular node.
C     This includes any available plans for subordinates.
C
C     PARAMETERS:
C     (INPUT)
C        PNode  ==> Pointer to current node
C        PStatQ ==> Pointer to top of queue of status blocks
C
      IMPLICIT INTEGER (P)
      INCLUDE 'Include.for'
      CHARACTER*17 TString
C
C     Output header lines.
C
      CALL ChTime (ITime, TString)
      WRITE(IOut, 900) TString
900   FORMAT(1X, 'PLAN UPDATE AT ', A17)

      WRITE(IOut, 910) (Memory(PNode+Pos), Pos=12,14)
910   FORMAT(1X, 'Plan for ', 3A4)

      WRITE(IOut, 920)
920   FORMAT(1X,13X,'½',9X,'GOAL',18X,'½',12X,'PERIODIC ',
     -   REVIEW', 12X,'½',4X,'FORCE RATIO',4X,'½')

      WRITE(IOut, 930)
930   FORMAT(1X,5X,'UNIT',5X,'½FORCE/RATIO½',8X,'TIME',7X,
     -   '½',7X,'START',7X,'½',7X,'DELTA',7X,'½',3X,'LOW',
     -   3X,'½',3X,'HIGH  ½')
      WRITE(IOut, 940)
940   FORMAT(1X,105('_'))
C
C     Output plan for current unit.
C
      CALL PrtLine(Memory(PNode+25))
C
C     Output plan for each subordinate unit.
C
      PStatus = PStatQ
450   CONTINUE
      IF (PStatus.EQ.0) GO TO 500
         CALL PrtLine(Memory(PStatus+53))
         PStatus = Memory(PStatus)
         GO TO 450
500   CONTINUE
      RETURN
      END
```

```
      SUBROUTINE ReadPlan (PMsg)
C
CC    PURPOSE: Read in the additional data lines for an
CC    external message containing red plans.
CC
CC    PARAMETERS:
CC    (INPUT)
CC       PMsg     ==> Pointer to message the additional data
CC                    lines are for.
C
      IMPLICIT INTEGER (P)
      INTEGER IVal(4)
      REAL Val(3)
      INCLUDE 'Include.for'

      Memory(PMsg + 17) = 0
C
CC    Read in the first line of additional data.
C
90    CONTINUE
         READ(Inp, 100) IVal(1), Val(1), (IVal(J), J=2,4),
     -                  (Val(K), K=2,3), More
100      FORMAT(6X, I6, F7.3, 3I6, 2F7.3, I6)
         WRITE(IOUT, 110) IVal(1), Val(1), (IVal(J), J=2,4),
     -                    (Val(K), K=2,3), More
110      FORMAT(10X,'MESSAGE DATA',I6, F7.3, 3I6, 2F7.3, I6)
C
CC       Save data in plan block and add to list.
C
         CALL Gimme(PPlan, 21, Memory)
         Memory(PPlan) = Memory(PMsg+17)
         Memory(PMsg+17) = PPlan
         Memory(PPlan+1) = Memory(PMsg+3)
         Memory(PPlan+2) = Memory(PMsg+19)
         Memory(PPlan+3) = IVal(1)
         Store(PPlan+4) = Val(1)
         Memory(PPlan+5) = IVal(2)
         Memory(PPlan+6) = IVal(3)
         Memory(PPlan+7) = IVal(4)
         Store(PPlan+8) = Val(2)
         Store(PPlan+9) = Val(3)
C
CC    Check if there is another data line to read in.
C
      IF (More.NE.0) GO TO 90
      RETURN
      END
```

```
      SUBROUTINE RptAction(PNode, IOpStatO, IOpStatN,
     -                     GCheck, POMP, POut)
C
C     PURPOSE:  Send a message from a subordinate unit in
C     combat to its commander. Message contains the old and
C     new values for units combat operations status index.
C     This tells the commander that the unit strayed from
C     the plan and had to take corrective measures.
C
C     PARAMETERS:
C     (INPUT)
C        PNode     ==> Pointer to unit doing this process.
C        IOpStatO  ==> Old combat operations status index.
C        IOpStatN  ==> New combat operations status index.
C        GCheck    ==> Identifies whether this message is for
C                      a goal time check. 0 ==> no  1==> yes.
C        POMP      ==> Pointer to the process.
C        POut      ==> Pointer to the output message format.

      IMPLICIT INTEGER (P)
      INTEGER GCheck
      INCLUDE 'Include.for'
C
C     Create additional data block for message.
C
      CALL Gimme(PPERMsg, 21, Memory)
      Memory(PPERMsg) = 0
      Memory(PPERMsg + 1) = Memory(PNode + 1)
      Memory(PPERMsg + 2) = ITime
      Memory(PPERMsg + 3) = IOpStatO
      Memory(PPERMsg + 4) = IOpStatN
      Memory(PPERMsg + 5) = GCheck
C
C     Send message.
C
      CALL MsgOut(PNode, POMP, POut, PPERMsg, 21, 0)
C
C     Subroutine MsgOut makes a copy of the additional data
C     block to be linked with the message. Therefore, return
C     additional data block to free space.
C
      CALL Releas(PPERMsg, 21, Memory)
      RETURN
      END
```

```
      SUBROUTINE XChngPlan(PSource, PDest)
C
C     PURPOSE:  Replace the contents of plan block with the
C     contents of another plan block. DO NOT replace pointer
C     to the next plan block.
C
C     PARAMETERS:
C     (INPUT)
C        PSource ==> Pointer to source plan block.
C     (INPUT/OUTPUT)
C        PDest   ==> Pointer to destination plan block.
C
      IMPLICIT INTEGER (P)
      INCLUDE 'Include.for'

      DO 100, Loc = 1, 3, 1
         Memory(PDest+Loc) = Memory(PSource+Loc)
100   CONTINUE

      Store(PDest+4) = Store(PSource+4)

      DO 140, Loc = 5, 7, 1
         Memory(PDest+Loc) = Memory(PSource+Loc)
140   CONTINUE

      Store(PDest+8) = Store(PSource+8)
      Store(PDest+9) = Store(PSource+9)
      RETURN
      END
```