REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
05-2001	Ph.D. Dissertation	

4. TITLE AND SUBTITLE
Anticipatory Planning with Agents Using Genetic Algorithms and Simulation

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
John Mitchell Duval Hill

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

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

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Department of Electrical Engineering and Computer Science
West Point, NY 10996

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
A. Approved for public release; distribution is unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER (include area code)
					John M. D. Hill (979) 680-9347

ANTICIPATORY PLANNING WITH AGENTS USING GENETIC ALGORITHMS AND SIMULATION

A Dissertation

by

JOHN MITCHELL DUVAL HILL

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2001

Major Subject: Computer Science

ANTICIPATORY PLANNING WITH AGENTS

USING GENETIC ALGORITHMS AND SIMULATION

A Dissertation

by

JOHN MITCHELL DUVAL HILL

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Approved as to style and content by:

Udo W. Pooch
(Chair of Committee)

John Yen
(Member)

Riccardo Bettati
(Member)

Michael T. Longnecker
(Member)

Wei Zhao
(Head of Department)

May 2001

Major Subject: Computer Science

ABSTRACT

Anticipatory Planning with Agents Using Genetic Algorithms and Simulation.

(May 2001)

John Mitchell Duval Hill, B.S., United States Military Academy;

M.A., The University of Texas at Austin

Chair of Advisory Committee: Dr. Udo W. Pooch

The traditional Military Decision Making Process (MDMP) focuses on developing a few friendly Courses of Action (COAs) against the "most-likely and most-dangerous" enemy COAs. There is a well-known axiom that "No plan survives the first shot." This indicates that a branch has occurred during execution that was not included in the plan, forcing the human planners into reactive mode.

The military is capable of producing unprecedented amounts of battlefield information that could be used to better anticipate the flow of the battle. Military planners need a way to incorporate this continuous feed of battle information into the planning process so that they achieve and maintain "option dominance". A new approach to military operations, called *Anticipatory Planning and Adaptive Execution*, treats planning and execution as a tightly coupled, single process, and replaces reaction to events with anticipation of events.

This research develops the methodology for automating the Anticipatory Planning process. A prototype *Anticipatory Planning Support System* (*APSS*) has been designed and implemented to provide human planners with an interactive visual

development system using simulations to build *Plan Descriptions*. *Nodes* represent option points in the plan and *Branches* represent the transitions between them. As execution progresses the plan is continuously updated based on actual events. *Execution Monitors* are attached to *Nodes*, use forward simulation from the *Actual State* to derive *Anticipated States*, and compare them with the *Planned State* at the *Nodes*. The *Execution Monitors* recommend re-planning to the *Planning Executive*, which prioritizes planning to maintain a balance between anticipating as many future branches to the plan as possible and constraining the planning effort. The *Planning Executive* launches *Planners* that use a genetic algorithm and inference mechanisms to postulate and consider possible friendly and enemy actions, then produce significant, representative, *Branches*. For testing or training purposes, an external *Stimulator* uses a controlled *Plan Description* and a simulation to produce *Actual States* for use by the *APSS*. The primary goals of this implementation are to provide a common representation of the plan, facilitate the planning process, anticipate the flow of the battle, and provide a means for stimulating planning systems.

DEDICATION

This dissertation is dedicated to my mother, Judith Williams Hill.

You shared your life with my father, you gave me my life, and you live in the memory of your sons.

I once explained to you how sad I was that my father did not live long enough to see me earn my master's degree and be promoted to Major. I had so much wanted to make him proud of me. You replied, simply, "He knew you would, and he always was."

Since you passed away, I have been promoted to Lieutenant Colonel and earned my Ph.D. I still want to make you proud of me. It comforts me to think that you know, and that you are.

ACKNOWLEDGEMENTS

I would like to acknowledge my advisor, Dr. Udo Pooch, for simultaneously challenging and supporting me throughout my doctoral program.

I also want to acknowledge my fellow doctoral students, John R. "Buck" Surdu, Curtis A. Carver, Jr, Michael S. Miller, Jeffrey W. Humphries, and James A. Vaglia, for the white-board and sounding-board work that made all of our efforts better.

Most importantly, I want to acknowledge and thank my wife, Margaret Lee Duval Hill, without whose love and support this dissertation would never have been finished. Thank you for loving me and sharing your life with me.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

In preparing for battle I have always found that plans are useless, but planning is indispensable.
– Dwight D. Eisenhower

A. Motivation

General (ret.) Wass de Czege has proposed a radically new approach to military planning and execution, which he calls Anticipatory Planning [1]. There are two main thrusts of the General's proposal. The first is that planning and execution should be treated as a tightly coupled, single process, rather than as distinct events. The second is that Anticipatory Planning is necessary in a dynamic and information-rich battlefield environment of the future.

In the traditional Military Decision Making Process (MDMP) the intelligence officers posit various enemy courses of action (COAs), and the operations and planning officers propose various friendly COAs to counter them [2]. Each of these friendly COAs are war-gamed in order to determine their viability. A COA is viable if it is suitable, feasible, and acceptable. *Suitable* means the COA accomplishes the mission and complies with the commander's guidance. *Feasible* means that constraints of available time, space, and resources are met. *Acceptable* means that the tactical or

---

This dissertation follows the style and format of *IEEE Automatic Control Systems*.

operational advantage gained justifies the cost in resources, especially casualties. Commanders often describe viability concerns in terms of desired end-state conditions at the conclusion of execution. The result of this analysis is a single, chosen COA for use in execution.

There is a well-known axiom that the plan never survives the first shot, which is another way of saying that a branch that was not considered in planning has occurred in execution. Consequently, the commander and staff are forced into a reactive planning mode. Rather than a long detailed plan relating stemming from comparisons of complete friendly and enemy COAs, the planners need a methodology that merges planning and execution. Such a methodology would develop and consider as many reasonable branches in the plan as possible in the initial planning process, and continuously update the plan as execution progresses. This coupling of planning and execution requires a new process.

The ability to develop and consider many branches in a plan necessitates an Anticipatory Planning process. Rather than choosing a single course of action and following it to conclusion, Anticipatory Planning involves maintaining as many possible friendly actions against as many enemy actions as possible. The plan is then considered to be a tree. The nodes of the tree represent states (i.e., snapshots of actual or predicted dispositions of forces on the battlefield) and are option points in the plan. The branches represent the transition to a node based on a particular enemy and friendly actions.

As new branches are developed, the Anticipatory Planning process continues planning ahead of the most likely branches. In this way, Anticipatory Planning for a

branch can be done well in advance and many options are kept open as long as possible, rather than reactive planning once the branch occurs. Anticipatory Planning will increase the importance of the information collection plan to quickly confirm or deny the viability of branches. One of the primary goals of the Anticipatory Planning process is to restrict the planning effort within system resources along these more likely paths.

New concepts coming out of the Information Technology Operations Center (ITOC) at West Point, NY, indicate that information operations (IO) stand on three legs, not two: Offensive IO, defensive IO, and Information Efficacy [3]. This research is designed to address the third. That is, how military planners can make the best use of the information that is available.

B. Research Objectives

The purpose of this research is to develop a methodology to support anticipatory planning during execution of military operations. To accomplish this, the following objectives must be met:

- Development of a Plan Description mechanism that facilitates control of the system by a Planning Executive, rapid simulation of entity interactions, and inspection and modification by Planning Agents.
- Development of Agents to Perform the required functions of Anticipatory Planning: Execution Monitor Agents, Planning Agents, Planning Executive Agent

- Development of the Anticipatory Planning Support System (APSS) methodology.
- Development of a prototype system that validates the Plan Description mechanism and the Planning Agents and demonstrates the feasibility of the APSS methodology.

C. Overview

A literature review of the relevant domains is presented in Chapter II. Some of the topics covered include military operations, simulation, artificial intelligence techniques including genetic algorithms, artificial intelligence planning, planning under uncertainty, and military modeling, simulation, and planning. Chapter III contains detailed descriptions of the anticipatory planning methodology and the design of the prototype system. This is followed in Chapter IV with details of the implementation. The experimental design, the tests used to verify and validate the methodology, and the results and analysis appear in Chapter V. To tie it all together, Chapter VI provides a summary of the research and conclusions drawn from the experiments.

CHAPTER II

LITERATURE REVIEW

A. Introduction

The major purpose of pursuing research into "A Methodology to Support Anticipatory Planning" is to provide a useful mechanism to aid in the planning, execution, rehearsal, and assessment of military operations. Military planning occurs in an uncertain environment, where an adversary is actively trying to defeat the plan. It is worthwhile to review several different areas of computer science that might contribute to the problem.

Initially, the military planning process is described. This provides the framework for discussion.

One key component of the Anticipatory Planning Support System will be the simulation used to predict anticipated states. Development of the simulation will also drive the test environment that will be used in the proof of concept. A review of simulation is included to examine some of the tools and techniques available.

It is worthwhile to consider a few artificial intelligence (AI) techniques, including adversarial search, and how AI has been applied to planning in general, planning under uncertainty, and finally in adversarial and military planning. This chapter provides a review of the literature and previous work in all of the above areas.

The Adversarial Search section covers many of the methods developed to determine optimal or acceptable game strategies. Although these strategies apply more

directly to two-player, turn-taking, games, they provide some interesting insights that may be very useful in adversarial and military planning.

The Artificial Intelligence Planning section covers some of the fundamental principles used in planning, and discuss the contributions made by several implementations. This information provides a useful frame of reference for the following sections.

The Planning Under Uncertainty sections cover several of the techniques used to deal with planning where actions can yield unexpected results in the environment, or the environment can change dynamically.

The Adversarial and Military Planning section describes previous work in planning in adversarial domains and in the specific domain of military planning.

This chapter concludes with a review of Military Modeling, Simulation, and Planning to provide more context for this research effort

B. Military Operations

The Army operations spectrum begins with planning, which is encoded in the Military Decision Making Process. When time permits, contingency plans are produced to account for possible branches in the main plan. Rehearsals are a mechanism for making sure subordinate units understands the plan. Execution of the operation may or may not follow the plan or the contingency plans. Hopefully, participants can understand and react appropriately to deviations from the plan. As soon as possible after execution, the unit employs an assessment process using after-action reviews (AARs) to

determine how well it performed and how to train in the future. This assessment includes an analysis of what the plan was, how conditions and enemy actions agreed or diverged from the plan, and how the unit reacted.

1. Planning

The United States Army uses the Military Decision Making Process (MDMP) to create an Operation Order (OPORD) that synchronizes the efforts of subordinate units to accomplish a given mission [2]. The MDMP has the following phases: *Mission Analysis*, *Course of Action (COA) Development*, *COA Analysis*, *COA Comparison*, *COA Approval*, and *Orders Production*. The purpose of this lock-step approach to decision making is two-fold. First, when followed correctly, it enables staffs to produce acceptable plans. Second, it ensures that every staff member follows the same process, even if they are inexperienced individually or as a staff team. This discussion will not attempt to teach the entire MDMP. Rather, it will focus on the steps of the process that are relevant to this project. Italics indicate words or phrases that are common military terminology.

a. Mission Analysis

The staff analyzes the higher OPORD to extract *specified tasks* and *implied tasks* (tasks that are not explicitly stated but are required to be performed nonetheless). From these, the staff identifies the *essential tasks* for successful mission accomplishment. With all of these tasks in mind, they determine the precise mission statement to use within their own order. This *restated mission* identifies the task the unit is to perform,

the purpose for accomplishing the task, the initiation or completion time and the location where the task will be accomplished.

Sometimes, having the right information at the right time is critically important to the commander. *Commander's Critical Information Requirements* (CCIR) help the commander visualize the flow of the battle and make timely decisions. They also help the staff filter the enormous amount of information that is developed during execution. The CCIR are broken down into three areas.

Priority Intelligence Requirements (PIR) identify information the commander needs to know about the enemy. PIR often help determine which COA the enemy has chosen or to ensure that the enemy is at the right place and time for a planned event, such as a counter-attack.

Essential Elements of Friendly Information (EEFI) identify information about friendly forces that the commander wants to deny to the enemy (they are sometimes referred to as "enemy PIR"). EEFI allow friendly units to place a priority on protecting or hiding specific information from the enemy.

Friendly Forces Information Requirements (FFIR) identify information about the status of friendly forces that the commander needs to know. For example, one element of the FFIR may be whether the status of the designated reserve force falls below seventy percent strength, which might preclude it from being committed. The staff selects an initial set of CCIR in the mission analysis phase.

The staff intelligence officer prepares an *Intelligence Preparation of the Battlefield* (IPB) that identifies and prioritizes possible enemy COAs. The staff will

typically focus on the enemy COAs that the intelligence officer has identified as *most likely* and *most dangerous.*

The task list, *restated mission*, CCIR, and probable enemy COAs are some of the inputs into the *COA Development* phase.

b. Course of Action Development

A critical part of the MDMP process is the *Course of Action Development* phase. During this phase the staff develops several candidate COAs based on the mission, the suspected enemy COAs, and the commander's guidance. The commander focuses the staff's efforts to produce several good COAs in the available time.

The candidate COAs must satisfy several criteria to be considered valid. Among these criteria are *Suitability, Feasibility,* and *Acceptability.* A course of action is suitable if it accomplishes the mission and complies with the commander's guidance. It is feasible if it accomplishes the mission within the constraints of available time, space, and resources. It is acceptable if the tactical or operational advantage gained justifies the cost in resources, especially casualties. Commanders often describe acceptability in terms of *desired end-state* conditions at the conclusion of execution.

The staff analyzes relative combat power to identify enemy vulnerabilities and determine where friendly capabilities can be applied against them. From this they develop a number of possible operations for the friendly and enemy forces. The staff usually considers force ratios, wherein friendly and enemy strengths are manipulated in a historically based mathematical model.

The staff considers the enemy COAs from most likely to least likely, or in an order specified by the commander. They use a brainstorming process to prepare friendly COAs that are capable of defeating each enemy COA. The commander and every member of the staff must ensure that they retain a *common picture* of the COA, and that the synchronization of effects is apparent to everyone. The staff members remain receptive to all ideas and counter-arguments to ensure valid COAs rise to the surface and invalid COAs are quickly discarded.

Within each COA, the *critical events* that ensure accomplishment of the mission are identified. Similarly, *decision points* where tactical decisions are required are noted. Decision points are linked to *Named Areas of Interest* (NAIs) that focus assets on gathering information necessary in order to make the decision.

For each COA, the staff designates a *main effort* and *supporting efforts*. They also organize the subordinate forces and assign *tactical tasks* for them to perform. The staff considers subordinate elements two levels below (e.g., a battalion staff considers the employment of platoons), arrays them to accomplish the tasks of the main and supporting efforts, and ensures each task has the right amount and mix of capabilities to succeed. As the staff refines the array of forces, they determine the *concept of the operation*. They incorporate and synchronize elements from all of the *battlefield functions* (maneuver, fire support, mobility / counter-mobility / survivability, etc.). They also add control measures, such as phase lines, where necessary to constrain the flow of the operation. The staff then assigns headquarters to be responsible for each of the tasks. The normal *span of control* for each headquarters is two to five subordinate elements.

COA development concludes with the preparation of *COA statements and sketches.* The COA statement describes the scheme of maneuver and states the subordinate tasks. The COA sketch depicts the maneuver and the control measures.

c. Course of Action Analysis (War-gaming)

In this phase the candidate courses of action are analyzed through a "war-gaming" process to determine if they are valid. As the war-game progresses, representatives from each of the battlefield functions (maneuver, fire support, etc) provide input about the expected results as the friendly courses of action are played out against the enemy courses of action. The staff employs an *action – reaction – counter-action* drill to describe the flow of the enemy and friendly COAs against each other. This phase ensures that everyone has the same understanding of the COA, that all resource requirements are identified, that all CCIR have been identified, and that all subordinate units and combat effects are synchronized. The results from each valid course of action will be used later in the *Course of Action Comparison* phase.

d. Course of Action Comparison

If the commander decides to war-game only one COA, or if he chooses one during the war game, no course of action comparison is needed. If multiple COAs have been war-gamed and the commander has not made a decision, the staff must conduct a COA comparison to aid the commander in choosing the "best" COA. All staff member state their findings so that they can be considered by everyone. The staff then uses a set of criteria appropriate to the mission, which may include weights, to compare the valid courses of action that survived the war-gaming.

e. Course of Action Approval

After reviewing the COAs and receiving the recommendations of the staff, the decides which COA to implement, along with any final refinement. He may also reject them and give the staff new guidance. He may even give the staff a completely different COA, possibly including components from several of the COAs the staff developed. In the latter two cases, the MDMP must be reiterated to ensure validity of the COA and synchronization. Once the commander makes his decision the staff issues a warning order so that subordinate elements can improve their planning.

f. Orders Production.

The staff takes the COA statement and sketch, and the results of the war-game for that COA, and refines them into a full-fledged operations order (OPORD). The OPORD give the subordinate units all the information the need for planning and execution. Before the staff issues the order, the commander reviews it and approves it one last time.

2. Rehearsal

Army units use rehearsals to help subordinate leaders visualize the flow of the operation and how it is synchronized. Depending on the amount of time available to the unit, rehearsals may be done over the radio, on a map, on a terrain model, even with the actual vehicles and personnel. Typically, the intermediate type of rehearsal on a terrain model is used. The staff and the subordinate leaders focus on the critical events and on the synchronization to ensure everybody understands how their unit fits into the plan.

3. Execution

During the execution of the operation the staff monitors the progress of the battle, directs activities to support the units, and works to keep everything synchronized. They look closely at information that indicates divergence from the plan, analyze it, and make recommendations to the commander. If all goes well, the staff and the subordinate units will keep the plan on track. Often, however, the actual battle diverges too far from the plan, and requires rapid analysis and decision-making to ensure a favorable outcome under the new conditions.

4. Assessment

The Army uses a well-defined After-Action Review (AAR) process to help units identify their strengths and focus their training to address deficiencies [4]. AARs are conducted during or immediately after the event to ensure everything is fresh in the participants' minds. They focus on the training objectives on performance against those objectives. Where possible, they include all available information about what the opposing forces were trying to accomplish. Key issues are discussed, and the relevant doctrine and tactics are reviewed. This self-assessment process is even more effective when some level of "playback" is available, such as video and audio of the event, or visualization.

C. Simulation

Simulation, like any other field of endeavor, has very specific terms associated with it. One of the key processes in using simulations is the building of models. There

are many advantages to using simulation. There are also some disadvantages. Finally, some techniques used in simulation, particularly "online simulation," are discussed.

1. Simulation Terminology

For clarity, the simulation terminology is broken into the following sections: characteristics of the model, environment, system, and the simulation. This organizational structure is suggested by Pooch [5].

a. Characteristic of the Model

An *entity* is "a real-world object" [5] or "an object of interest in the system" [6]. An *attribute* is a characteristic or property of an entity [5].

An *event* is an "instantaneous occurrence that may change the state of the system" [6]. Another way to look at it is that if the state of the system has not changed, no event has occurred [5].

An *activity* is defined as "any process that causes changes in a system" [5]. The distinction between an event an activity is that "an *activity* is like an event, but it occurs over some length of time, rather than at an instant in time" [7].

A simple definition of the *state* of a system is "a description of all the entities, attributes, and activities, as they exist at some point in time" [5]. A more precise definition is that a *state* is the "minimal collection of information with which the future state can be uniquely predicted in the absence of chance events" [5].

b. Characteristics of the Environment

They *system environment* is defined as "the objects and processes (entities and activities) surrounding the system" [5].

Endogenous activities are "activities that occur within the system" [5]. *Exogenous Activities* are "activities in the environment that affect the system" [5]. "The classification of all activities as either endogenous or exogenous establishes the *system boundary*" [5].

"A system with no exogenous activities is called a *closed* system; otherwise the system is *open*" [5]. In an open system, the state of the system changes in response to both endogenous and exogenous activities [7]. In a closed system, all state changes are driven by endogenous activities.

c. Characteristics of the System

Law and Kelton asserted, "Few systems in practice are wholly discrete or continuous, but since one type of change predominates for most systems, it will usually be possible to classify a system as being either discrete or continuous" [8].

"*Continuous* systems include variables that can assume any real value in a prescribed set of intervals" [5]. Continuous simulations are those in which parameters can be described by a series of differential equations [9].

"*Discrete* systems include variables that can assume only particular values from among a finite set of alternatives; these systems are characterized by discontinuous changes in the system state" [5] A discrete simulation is one in which the state variables change instantaneously only at discrete sets of points in time [6]. An example of such a system is Automated Teller Machine, in which a transaction happens instantaneously [7].

A *deterministic* system is one in which the next state of the system is completely determined by the current state and some event or activity. An example of this is a finite state machine [10].

In a *stochastic* system, there is some degree of randomness in the system. In a stochastic simulation, given the current state and some activity, the next state will be one of many possible states. Known families of probability distributions usually characterize the randomness in the system, but in some cases it may be possible to assign exact probabilities to each state transition [7].

A *static* system is one in which the state of the system is independent of time. A *dynamic* simulation is one in which the state of the system changes over time. Examples of dynamic simulations are those that describe movement of parts through a manufacturing facility, flow of electrons from a nuclear explosion, or attrition of combat forces during a battle [7].

Table 1: Validation Techniques

Validation Techniques from Law and Kelton	
Technique 1	Develop a model with high face validity
Technique 2	Test the assumptions of the model empirically
Technique 3	Determine how representative the simulation output data are

d. Characteristics of the Simulation

Validation "refers to the proof that the model is a correct representation of the real system" [5]. Law and Kelton provide a three-step approach (extending earlier work by Naylor and Finger [11]) for developing valid models [8]. The steps are shown in Table 1.

Verification "refers to the proof that the simulation program is a faithful representation of the system model" [5]. Law and Kelton describe several techniques for verifying that the simulation system, or the entire system, performs as designed [8]. The verification techniques are listed in Table 2.

Table 2: Verification Techniques

Verification Techniques from Law and Kelton	
Technique 1	Write and debug the program in modules or subprograms
Technique 2	Have multiple people review the programming
Technique 3	Run the simulation under a variety of settings of the input parameters and see if the output is reasonable
Technique 4	Print out traces or use an interactive debugger to ensure each step in the system is performed correctly
Technique 5	Run the model under simplifying assumptions
Technique 6	Observe an animation of the simulation output
Technique 7	Compare distributions produced by the random elements of the system with the desired distributions
Technique 8	Use a simulation package to reduce the programming effort, but be mindful of built-in errors and inefficiencies

Banks, et al., provide a substantially similar list [12]. However, they recommend graphical interfaces for accomplishing verification and validation, citing its usefulness as a form of self-documentation (due to Bortscheller and Saulnier [13]).

Experimental Design "refers to a sequence of simulation runs in which parameters are varied, with both economy and sound statistical methodology considered in achieving some specified goal" [5].

A *terminating* simulation is one for which "there is a 'natural' event E that specifies the length of each run (replication)" [8]. A system could be set to terminate at a specified (simulation) time. In simulations of adversarial situations, termination may occur when one side wins. In a terminating simulation, "*since the initial conditions... generally affect the desired measures of performance*, these conditions should be representative of those for the actual system" [8].

A *non-terminating* simulation is one in which there is no natural event E at which time the simulation run should stop [8]. The output of such a simulation is the *steady-state* value of some output parameter.

The *warm-up period* refers to the period when the system is affected by the initial conditions before reaching a steady state [5].

A *steady state* is reached when "successive system performance measurements are statistically indistinguishable" [5].

2. Building Models

One of the most important processes in building a simulation system is to correctly model the system being investigated. A system model is a representation of the

real system using specific information gathered for the purpose of studying the system. Pooch describes several different types of models: descriptive, physical, mathematical, flowcharts, schematics, and computer programs [5].

Abstraction is an important concept in modeling [14]. Zeigler notes that abstraction is "the process underlying model construction whereby a relatively sparse set of entities is extracted from a complex reality" [15]. In other words, abstraction helps the analyst focus on the desired level of detail of the system being modeled. Sisti says abstraction is "the intelligent capture of the essence of the behavior of a model without all the details (and therefore runtime complexities) of how that behavior is implemented" [16].

3. Advantages of Using Simulation

There are many advantages to using simulation. Adkins and Pooch identify the following advantages: controlled experimentation, time compression, sensitivity analysis by manipulation of input variables, no disturbance of the real system, and it is an effective training tool [5, 17]. Banks adds several other advantages, including helping people to make correct choices, diagnose problems, identify constraints, and specify requirements [18]. Some of these advantages are examined below.

One really good reason for using simulation is that it might be impractical or impossible to experiment with the real-world system. It is often preferable to conduct simulation experiments beforehand to determine behaviors, requirements, expected throughput, or other characteristics of a system.. This is particularly important when a system is under design. Carson and Banks define a simulation as the "imitation of the

operation of a real-world process or system over time," and this imitation provides a mean for experimentation [6].

It is particularly impractical to experiment with military operations. For an extreme example, it is "impractical to obliterate much of the surface of the Earth in order to explore the effects of nuclear war" [7]. This is why simulation has become so important to the military for training..

One of the key advantages of using simulation is that it helps people understand complex systems. Hoeber states that models should always "shed light," since the process of constructing the model should increase the understanding of the system by both the model builder and the client [19]. The purpose of simulation is to provide a tool with which to experiment to "gain some understanding of how a real system behaves" [8]. Simulation is often used to evaluate a model numerically, gather data, and estimate the true characteristics of the model [8].

Simulation also helps decision makers in choosing correctly. Hoeber asserts that modeling can aid in making choices since the decision maker will have a better idea of the possible outcomes [19]. Pooch notes that simulation allows an analyst to compare strategies for future operation of the system [5].

4. Disadvantages of Using Simulation

There are also some disadvantages to using simulation. Adkins and Pooch note that creating a simulation model can be expensive in terms of manpower and computer time, extensive development time may be encountered, hidden critical assumptions may cause the model to diverge from reality, and model parameters may be difficult to

initialize [5, 17]. Banks adds the following disadvantages: model building requires special training, simulation results may be difficult to interpret (or to explain), and simulations may be used inappropriately [18]. Some of these disadvantages are explored below. Clearly, cost-benefit analyses and tradeoff considerations must occur before analysts choose a simulation approach.

One of the major disadvantages to using simulation is that it may be used inappropriately. For instance, there are many situations where an analytical solution is more appropriate [18]. Another occurs when a single simulation run (experiment) is used to make decisions [7]. In stochastic simulations, many experiments must be performed in order to gather statistically valid data on which to base decisions.

Law and Kelton asserted that in many cases ten to fifteen experiments for a given set of parameters and initial conditions is sufficient [8]. While this number seems small, the reason is the high degree of uncertainty in the creation of the model. Given any large, complex simulation, the probability distributions used to estimate various parameters have some degree of error associated with them [7]. Running hundreds of simulation experiments will decrease the size of the confidence interval around output parameters, but Law and Kelton asserted that this gives a false sense of precision [8].

5. Discrete Event Simulation

Discrete event simulation is a technique used when system events occur at specific times and there is no concern for the interim periods between events. Or, as Law and Kelton describe it, discrete event simulation "concerns the modeling of a system as it evolves over time by a representation in which the state variables change

instantaneously at separate points in time" [8]. Events are stored in an event queue and are executed in order. As each event executes it may produce more events for the future. These events are placed in the correct order in the event queue for execution at the appropriate time.

In a discrete event simulation, time is normally moved forward as each event is removed from the ordered event queue. Another method, appropriate when the simulation is used to represent the actual time progress of a system, is to advance the clock by uniform steps, executing the events when their time arrives. The time-step method is simple, but trying to determine the correct size of the time step can cause problems, both in designing the simulation and producing the desired outputs [9].

Another consideration in discrete event simulations that must be addressed is how the simulation will be driven. One way to do this is to feed in historical data from the actual system. This serves only to validate the simulation. In order to perform analysis, the inputs must be modeled and must be configurable [20, 21].

6. On-Line Simulation

As computing and simulation technology have advanced, the ability to perform simulation on-line has been improved. The idea is to have a simulation of the system "thinking ahead" of the actual system. The discussion in this section is due to Surdu [7].

Davis discusses the difficulties in using offline simulations for performance improvement and proposes "on-line" simulation as a method for the improving the performance of real-time systems [21-23]. In addition to Davis' work, Andersson and Olsson proposed using simulation in a customer-order-driven assembly line [24]. In the

military arena, Ferren discusses how the warfighter can use simulations as a predictive tool embedded in all manner of military systems [25].

There are two significant concepts that come out of this work in on-line simulation [21]. First, the *control policy* provides a means of directing changes to input parameters in response to conditions within the system that will move the system towards improved performance. The second concept is *autovalidation*, where the results of the on-line simulation are compared periodically to the operation of the actual simulation, enabling modification to bring the simulation closer to the actual system. Surdu implemented an autovalidation mechanism in the OpSim project [7].

7. Simulation in Java

McNab and Howell built a discrete event simulation library in Java, based on an earlier SIM++ library for C++ [26]. Their main purpose in doing so was to enable easy building and display of simulations through the World Wide Web. In the process of building simulations based on the library they quantified some of the implementation and performance implications of using Java.

D. Artificial Intelligence Techniques

There are three areas in Artificial Intelligence that seem like they will be particularly applicable to this research project. Since the project will include the examination of a large search space to determine the most-fit solutions, a review of genetic algorithms is appropriate. There are also situations where given particular conditions an exact outcome is desired. Crisp inference systems provide that capability,

and are examined. Although the adversarial environment being examined in this research is not the same as AI adversarial search, an examination of those techniques reveals some insights into the work for this project.

1. Genetic Algorithms

Genetic Algorithms (GAs) draw on the adaptive "survival of the fittest" capabilities inherent in Darwinian evolution. One fundamental aspect of a GA is an encoding that allows the description of every possible state of a system, but which is also amenable to rapid calculation. This encoding is typically referred to as the "chromosome," although the term "genome" may be appropriate if the encoding contains distinguishable sub-sections. Another fundamental piece is a "fitness function" which is used to decide how good the outcome of the system is when a particular chromosome is used. The algorithm creates an initial population of the chromosomes, possibly using heuristics to ensure a pretty good set. The fitness function is applied to each chromosome, allowing them to be ranked. As the algorithm produces each new generation, the more fit member of the previous generation have a higher probability of reproducing. Children for the new generation are produced by pairing two parents, and with some probability crossing their genes. Also, with a small probability, the children may experience a mutation in the elements of the chromosome. A seminal discussion of genetic algorithms appears in DeJong's dissertation [27]. Goldberg provides a thorough presentation of GAs in his book [28].

2. Crisp Inference

Rule-based systems allow knowledge to be represented as actions to be taken when certain conditions are matched. These heuristics, or "rules of thumb," are normally chosen by a domain expert and encoded by the developer. These rules allow abstract, symbolic approaches to be used in specifying knowledge based on human logic. CLIPS is a forward chaining LISP-like rule-based language that has inferencing and representation capabilities and is used to build rule-based expert systems [29]. CLIPS processes the rules by using RETE, an algorithm that solves the difficult many-to-many matching problem encountered when matching rules with facts [30].

3. Adversarial Search

This section will examine two-player games where both players know everything about the game (Two-Player Perfect Information Zero-Sum Games). Further, the players will take alternating turns, allowing the game to be represented as an AND/OR. The root node of the tree is the initial situation. The edges in the tree represent legal moves. Each level in the tree represents the possible moves for a player, and the next level is the moves for the opponent, alternating until conclusion. The idea is for the player to pick the sequence of moves that will move the game through the tree to a leaf node that concludes the game with a win [31-33].

a. MiniMax Algorithm

The minimax algorithm is a depth-first search algorithm that selects the best possible move for a player at each turn. Basically, minimax builds a tree representing the search space and assigns a score to the root node. This score is based on the

assumption that at each level of the tree the player will take the move that maximizes the score and the opponent will take the move that minimizes the score. For complex games that would result in very large game trees, the minimax algorithm may be given a depth bound. The final result may be sub-optimal in this case, but each individual move will be optimal with respect to the allowed depth [31, 33].

b. The Horizon Effect

The reason placing a depth limit on minimax makes it potentially sub-optimal is that a good sequence of moves may be masked from the player. This is known as the horizon effect. There are two proposed solutions to this problem, neither very satisfactory [32].

Secondary Search: The algorithm chooses the best move and violates the depth limit to look a few moves further down. The idea is ensure there is no sudden drop-off on the other side of the horizon if the best move is taken. [32]

The Killer Heuristic: This heuristic focuses on the opponent's possible moves and examines the possible sequences following a particularly good move. This helps the player determine whether to let the game proceed to a state where the opponent can select that move [32].

c. Solve

The Solve algorithm is an extension of the minimax algorithm that reduces the number of nodes expanded by ignoring nodes beneath a known winning node or beneath a known losing node. This is a pretty obvious heuristic that can significantly reduce the search space [33].

d. Alpha-Beta Pruning

Alpha-Beta pruning is based on the minimax algorithm, but keeps track of the value of each path generated so far. Two variables are kept during the search. The alpha value represents an upper bound for the outcome of the path and a beta value represents a lower bound for the path. At maximizing levels (when the player moves), only beta is used to cut off the search. At minimizing levels (when the opponent moves) only alpha is considered [32].

As each path is examined the outcome value is determined. This is compared to the alpha and beta values of the other paths. If the outcome value of the current path is lower than the alpha value, then the current path can be removed from consideration, since it will never be taken. If it is higher, the path represented by the alpha value can be discarded, and the alpha value adjusted to the value of the current path. Conversely, for opponent moves, if the outcome value of the current path is higher than the beta value of any other path, the current path can be removed. The alpha value and beta value are updated as higher and lower values, respectively, are discovered [31-33].

Russell and Norvig describe the development of alpha-beta search and several implementations [31].

- John McCarthy conceived the idea of alpha-beta search in 1956, although he did not publish it.
- Newell developed NSS in 1958. It was the first chess program to use a simplified version of alpha-beta [34]

- Arthur Samuel's checkers program also used alpha-beta, according to Nilsson [35], although Samuel did not mention it in the published reports on the system. [36, 37]
- Hart, *et al.*, described a Tree Pruning (TP) algorithm in 1961 [38]. The title was updated to "Alpha-Beta Pruning" in a 1963 revision. Brudno examined bounds and variations in alpha-beta pruning [39]
- Slagle examined game trees and reported on m & n minimaxing [40]. Slagle also implemented an alpha-beta based system to play kalah (a two-player game involving allocation of beads between several bowls) [41].
- Kotok used alpha-beta in the "Kotok-McCarthy" chess program [42].
- Greenblatt used alpha-beta in the MacHack 6 chess program, which was the first chess program to successfully compete with humans [43].
- Knuth and Moore reviewed the history of alpha-beta and provided a proof of its correctness and a time complexity analysis [44].
- Pearl conducted further analysis of the effective branching factor and time complexity of alpha-beta and showed that alpha-beta is asymptotically optimal among all game-searching algorithms [45].

e. B* Algorithm

Berliner describes the B* method, which can prove that a branch from the root of a search tree is better than all the others [46]. It uses a best-first strategy to determine the order of node expansion, and assigns optimistic and pessimistic bounds to each node. These bounds tend to converge, leading to a termination of the search at that point. In

this way, greater responsibility is given to the evaluation functions, which may be used against any property or set of properties of domain. Berliner provided experimental and analytic evidence that B* is a very effective method of searching adversary trees.

Palay has shown that representing the range of a node as a probability distribution considerably improves B*'s performance [47]. A distribution provides a more accurate assessment of what is in a sub tree than a range does. It also allows termination of a search based upon probabilistic criteria.

f. SSS* Algorithm

Stockman developed the SSS* algorithm in1979 [48]. This new approach traded storage space for the ability to keep track of several alternate search paths simultaneously. SSS* is a best-first search procedure that keeps upper bounds on the values of partially developed candidate strategies. The best candidate strategy is chosen for further exploration. When this process is complete one of the strategies has been fully developed and must be the optimal strategy [32].

g. SCOUT Algorithm

The SCOUT algorithm uses a test function to evaluate a node by computing the minimax value v of its first successor. It then "scouts" the remaining successors to see if any of them are better. It is faster to perform this test than to determine the minimax value of all the successors. Once the best successor has been determined in this fashion the value is passed back up in the algorithm [32].

h. Performance of Game Searching Algorithms

Doyle presents the argument that shows that every search strategy that evaluates a game tree must examine at least twice the square root of the number of nodes in the tree [32].

E. Artificial Intelligence Planning

This section is included to provide the foundation for the discussions of adversarial planning and planning under uncertainty to follow. As such, it covers the terms appropriate for AI planning and reviews some of the planning systems that form the foundation for AI planning

1. Terms

In any academic discussion, it is important to have a common vocabulary to ensure precise understanding of ideas when they are communicated. In the area of Artificial Intelligence Planning, the literature demonstrates a consensus among researchers as to the meanings of several commonly used terms. This section provides the common vocabulary appropriate to this paper. Many of the definitions are due to Doyle [49].

a. Linear versus Non-Linear Planning

In planning, the term "linear" means that the operators are independent of each other and can occur in any order. However, they must be executed one after the other. The plan can be represented as a single line from a node representing each action to the next action.

Non-linear planning allows actions to occur simultaneously. The plan can be represented as a directed graph or a network

b. Hierarchical versus Non-Hierarchical Planning

In planning, a hierarchical planner uses a hierarchy of abstractions to solve the plan. At the higher levels, tasks are more abstract; at the lower level they are more concrete. The purpose behind this system is to simplify planning by focusing on the more abstract levels to find workable plans, then working out the details.

A non-hierarchical planner uses tasks that are all at the same level. In non-hierarchical planning, the planner does not distinguish between more important goals and less critical ones, and can potentially waste a lot of effort on unimportant steps.

c. Backtracking

When an action threatens a pre-condition of another action and the threat cannot be resolved, the planner must backtrack to a state before one of the actions was decided on and attempt to find a different plan.

d. Early Commitment versus Least Commitment

In early commitment, the planner commits to an operation that satisfies a precondition as soon as it can be done. In a least commitment strategy, the planner delays committing to any particular operation until it has to. The idea is to prevent interference with past or future decisions, and to reduce the amount of backtracking required.

e. Planning under Uncertainty

Uncertainty occurs when there is no guarantee that an action will produce the post-conditions it is supposed to. From one perspective, this occurs when post-conditions have a probability of occurrence rather than a certainty. From a different perspective, planning can be uncertain if the environment can change even without actions taken by the planner. When planning under uncertainty, it is usually necessary to monitor the execution of the actions and to sense the state.

f. Execution Monitoring

Observing an action to determine if it produces the post-conditions it was supposed to.

g. Sensing

Sensing is deliberate gathering of information from the environment.

h. Adversarial Planning

Adversarial planning occurs when there is another player or agent that is actively trying to defeat the plan. This is an example of planning under uncertainty, since changes in state may occur based on the adversary's actions.

i. Military Planning

Military planning is by its very nature uncertain and adversarial. It is also very dynamic and includes considerations not present in most classical AI planning environments. One of these factors is the highly stochastic nature of the outcome of actions during execution. Another factor is the fact that the actors themselves can be

consumed. Actual military planning performed by humans is an art, supported by the science of procedures of logistics.

2. Artificial Intelligence Planning Systems

The application of Artificial Intelligence (AI) techniques to planning has been underway for quite some time. This section covers some of the fundamental principles used in planning and discusses the contributions made by several implementations. The identification of the systems is mostly due to Doyle [49]. Many of the systems and approaches are related to each other (a good diagram of the relationships appears in [50]). In some cases, approaches from the different systems have been integrated. Rather than try to follow all of the interconnections, the systems and approaches are present in chronological order of appearance in the literature.

a. GPS

Newell and Simon developed the General Problem Solver (GPS) as a research tool for examining human and artificial thought processes [51]. This system, presented in 1961, is an example of a hierarchical linear planner. GPS was intended to model human thought in solving search problems, and under specific assumptions, can be used to produce plans of action [52]. GPS uses means-end analysis in which the system uses operators to reduce the differences between the present state and the goal state.

b. STRIPS

The STRIPS planner developed by Fikes, et al., represented states as a world model and a set of goals to be achieved on a stack [53, 54]. This system was developed in 1971 and is an example of a non-hierarchical linear planner. Operators in the STRIPS

system have preconditions that must be met before execution. If an operator is used because its result satisfies a goal on the goal stack, then its preconditions became new goals on the stack. When all pre-conditions of an operator are satisfied, the operator can be executed, producing a new state. This new state is determined by executing the "add-list" and the "delete-list" (which originated with STRIPS) associated with the operator [52].

c. ABSTRIPS

Sacerdoti's ABSTRIPS planner extends STRIPS to consider levels of abstraction in which some operators are more critical than others [55]. ABSTRIPS, presented in 1974, is a hierarchical linear planner. Sacerdoti commented on the inability of the heuristics-based approach of STRIPS or GPS to solve reasonably complex problems, and proposed a means for determining between important information and mere details. Through the use of an abstraction hierarchy, problems can be solved at high-level of abstraction, and then the lower-level details can be worked out. If for some reason the details cannot be arranged correctly, re-planning can be performed at the higher levels. One advantage to this approach is that dead ends can be determined early and removed from consideration. Knobloch would later show in an analysis of ABSTRIPS that in cases where the independence assumption on preconditions did not hold, ABSTRIPS would actually degrade performance [56].

d. NOAH

Sacerdoti's Nets of Action Hierarchies (NOAH) system used a partial order to represent the structure of a plan and implemented a more elaborate set of ordering

constraints that could resolve different classes of conflicts [57]. NOAH, presented in 1975, is a hierarchical non-linear planner. NOAH uses a least-commitment strategy based on a hierarchy of abstract to concrete operators. This hierarchy is formed into what Sacerdoti calls "procedural nets." NOAH also has "critics" that examine the plan to resolve conflicts, eliminate redundant preconditions, and deal with unbound variables. NOAH suffers from the problem that it may commit to one of several constraints and is unable to backtrack to repair a failed plan.

e. NONLIN

Tate's non-linear planner (NONLIN) system, a hierarchical non-linear planner presented in 1977, fixed one of the problems with NOAH by providing facilities for backtracking [58]. NONLIN keeps a list of all decisions made and has plan-modification operators so that faulty plans can be repaired. If a prior decision blocks a necessary action later in the plan, the decision point is known and the system can backtrack to that point. Also, the alternative decision possibilities are known and a new one can be selected. NONLIN was later improved to use dependency directed backtracking [49].

f. MOLGEN

Stefik's molecular genetics (MOLGEN) system used a technique he called "constraint posting" to consider the interaction of sub-problems [59]. Introduced in 1981, MOLGEN is a hierarchical non-linear planner that imposes additional constraints on variable bindings to help resolve conflicts. MOLGEN does this by providing three layers of abstraction. Goal relations are handled in the strategy layer. The specifics of

the plan are handled in the planning layer. Constraints are dealt with in the design layer. Stefik refers to this layered control structure as meta-planning.

g. SIPE

Wilkins' System for Interactive Planning and Execution Monitoring (SIPE) system incorporated the ability to construct partial descriptions of planning variables that have not been instantiated [60, 61]. SIPE, presented in 1984, generates hierarchical, partially ordered plans. This partial description ability imposes additional constraints on variable bindings, helping to resolve conflicts. Put simply, when an operator requires a particular precondition it sets a flag on that precondition. If two operators that are not already ordered try to affect that precondition one or the other is promoted, removing the conflict. [49]

h. TWEAK

Chapman's TWEAK system introduced an additional type of constraint on variable bindings that forces two variables to instantiate to different objects [62]. Introduced in 1984, TWEAK is a non-hierarchical non-linear planner designed to address the "scruffy" nature of previous planners. The TWEAK system attempts to minimize backtracking by incrementally specifying constraints. The system only has to backtrack when a set of constraints becomes inconsistent. Chapman also provided a formal language for expressing plans. The TWEAK planner, within its design constraints, is provably correct and complete. This correctness relies on the idea of the "modal truth" criterion, where a step C that clobbers a proposition P can be followed by a step W which re-asserts P before it is required in later steps [49].

i. WATPLAN

Yang presented a theory for resolving conflicts after constraint-based plans have been generated [63]. His WATPLAN system, introduced in 1992, is an example of a hierarchical non-linear planner. Fundamentally, Yang's theory maintains a global perspective on conflicts requiring resolution, rather than resolving each conflict incrementally. By careful selection of which conflicts are resolved first, and by early detection of dead-end plans, less computation is required. The WATPLAN system implements this theory by representing conflicts as constrained variables and attempting to force a solution to the set of constraints, i.e., making it a formal constraint satisfaction problem (CSP).

F. Planning Under Uncertainty

Although there are many ways to view planning under uncertainty, there is an overview of approaches due to Olawski that is quite appropriate for this research project [64]. The following sections provide a review of some of the more notable ideas and systems developed for planning under uncertainty. The identification of some of the systems is due to Pryor [64] and to Russell [31]. Once again, a chronological presentation should suffice to demonstrate the important concepts and advances.

1. Overview

Most of the early planning systems operate in environments where the only changes occur in response to actions taken during execution of the plan. This allows the planner to develop a plan that will work in predictable ways when executed. In some

domains, however, there are external agents operating on the environment,. In addition, the outcomes of actions taken may be probabilistic. Generally, there are four approaches to solving the problem of planning under uncertainty [64].

In *contingency planning*, classical planning is extended to develop a single plan that will succeed in all circumstances. Of course, this can result in very large storage requirements, since every contingency must be accounted for.

In *probabilistic* or *decision-theoretic planning*, where the outcomes of actions are stochastic, the planner tries to construct a plan that has a high probability of succeeding. This reduces the storage requirement, since not all contingencies are considered, but is susceptible to failure since there is no guarantee that particular outcomes will occur.

Another approach is to *interleave planning and execution.* In this method, the plan is not developed fully in advance. Rather, the plan is developed based on what happens during execution. There are different strategies for merging planning and execution, but they all have some drawbacks. Execution monitoring and sensing are usually very important in these approaches.

Finally, in *reactive planning*, the behavior of the planner is controlled by a set of reaction rules. Rather than attempt to account for each contingency or try to develop a probably successful plan, this method develops a set of rules to guide selection of the actions to be taken. This approach is useful in some cases, but often suffers from insufficiently specified rules. Execution monitoring and sensing is important in this approach as well.

2. Planning Under Uncertainty – The 1970s

a. PLANEX

Fikes, et al., presented the planning and executing system PLANEX in 1972 [54]. This system worked with the STRIPS planner to control the actions of the Shakey robot, and was the first major treatment of execution monitoring. PLANEX used triangle tables to allow recovery from partial execution failure without having to completely re-plan [31].

b. Hacker

Sussman developed a system called Hacker in 1973 that applied ordering constraints called "hacks" on the operators of a plan [65]. This is also a non-hierarchical linear planner. The basic idea is to create a plan and then repair it. After a plan is generated it is examined for known conflicts. If a known conflict is found it is resolved by an associated hack. If a previously unknown conflict is discovered, a new hack is created. The addition of new hacks is how the Hacker system acquires new planning skills.

c. WARPLAN

Warren developed an early contingency planner called, simply, WARPLAN, in 1974 [66, 67]. WARPLAN-C, a variant presented in 1976, was a small case-based reasoning system that used promotion and backtracking to prevent problems like the Sussman Anomaly [49]. This planner was based on predicate calculus rather than a STRIPS-style action representation and was limited in the number of possible outcomes

[64]. Each conditional could only have two outcomes, true or false. If the first led to failure, the second outcome was chosen [68].

d. NASL

McDermott presented the NASL planner in 1978 with the idea of embedding problem solving in the theory of action, making a "problem" just an action that cannot currently be accomplished [69]. In this way, planning and execution were completely unified [31]. Although not provably complete, the NASL implementation did make progress towards McDermott's goals of "analytical and heuristic adequacy."

3. Planning Under Uncertainty – the 1980s

a. DEVISER

Vere added the ability to consider operators and goals that have time windows associated with them in the DEVISER system [70]. Introduced in 1983, DEVISER is a hierarchical non-linear planner. By using time windows, parallel tasks that have "no earlier than" and "no later than" considerations can be handled. DEVISER also handles resource consumption during execution of the plan.

b. PRS

Goergeff and Lansky developed the Procedural Reasoning System (PRS) in 1986 to incorporate belief, desire, and intention in the planner [71]. Actions are taken based on current desires or goals, beliefs about the environment, and current intentions. This avoids overly strong expectations about the environment, overly constrained plans of action, and other forms of over-commitment. One significant effect of this approach is

that a current plan can be interrupted to handle a higher priority problem, or the current plan can be completely abandoned when beliefs, desires, or intentions change.

c. PENGI

Chapman and Agre presented the PENGI game-playing system in 1987 [72, 73]. Chapman reported the theoretical difficulties with planning and the inadequacies of the symbolic AI model. Agre observed that most activity is 'routine' and requires little new abstract reasoning. He proposed the idea that most routine decisions can be encoded into a low-level structure that only needs periodic updating. This approach was implemented in the PENGI system [74].

d. Universal Plans

In 1987, Schoppers presented the Universal Plans approach to reactive planning [75]. Schoppers notes that PRS deals with the *means* to achieve goals, but does not examine situation-dependent adoption and abandonment of goals. His idea is to build a goal-directed "universal plan" that can produce appropriate behavior in unpredictable environments. This universality is gained by producing a number of reaction rules that describe what to do if a particular condition occurs. Russell notes that this approach is really just a rediscovery of the idea of policies in Markov decision processes [31]

e. IPEM

Ambros-Ingerson and Steel developed the integrated planning, execution, and monitoring (IPEM) system in 1988 [76]. This was the first system to smoothly integrate partial-order planning and planning execution [31]. IPEM operates on the principle that steps are only executed when no further planning is possible. If execution of a step

enables further planning, the new planning will be exhausted before the next step is taken. Two later planners, XII [77] in 1994 and Sage [78] in 1995, will use the same operating principle [64].

f. ADL Representation

Pednault described the Action Description Language (ADL) in 1989 [79]. ADL is syntactically similar to STRIPS, but allows for a more powerful specification of preconditions and effects. One advantage of this representation is that it is domain-independent and allows the knowledge base to be built dynamically from the domain, rather than having the knowledge hard-coded as actions [80]. As such, it is well suited as a foundation for planners that operate under uncertainty.

g. Situated Control Rules

Drummond presented a two-stage analysis in 1989 that synthesized situated control rules (SCRs) [81]. Drawing on Schopper's universal rules, SCRs characterize the performance of possible actions by an agent based on its current environment. The idea is that both an executor, which runs the plan, and a projector, which analyzes the plan, accepts a plan net. The projector produces the SCRs based on predicted situations, and the executor checks to see if any SCRs have been developed for the current situation.

h. BUMP

Olawsky and Gini investigated the effects of different plan and execution interleaving strategies in designing their basic University of Minnesota Planner (BUMP) system in 1989 [82]. The authors examine three approaches for incorporating sensing into planning. One approach is to plan for all contingencies, that is determine all

possible sensing results and plan for that eventuality. This is very expensive in terms of storage of the plan space and processing time. Another approach is to form a complete plan based on an assumed value of the sensor reading. This strategy is less expensive, but if any of the assumed sensor readings are incorrect, the plan will most likely be invalid and require re-planning. The third approach is to defer planning decisions that require sensor information until the information is available. This approach avoids some planning that would end up discarded, but may require some actions that were already executed to be undone. If those actions are not reversible, the plan may be invalid.

The BUMP system focuses on the third approach, and the authors develop two strategies for deferred planning. In the Continue Elsewhere strategy as much preplanning as possible was performed. In the Stop and Execute strategy, goals defined in terms of sensor readings were executed as soon as they were encountered. Neither strategy was shown to be better than the other, since both strategies sometimes produced invalid plans. [64]

i. O-PLAN

Tate introduced the open planning (O-PLAN) architecture in 1989 [83], and Currie and Tate described it in great detail in 1991[84]. The O-PLAN architecture started out as a derivative of NONLIN used to support research and development into planning systems, with a focus on coordinating planning and execution effort. It uses a mixture of artificial intelligence techniques and numerical techniques from operations research. O-PLAN uses a task formalism (TF) to describe the domains in which it is asked to operate. The main contributions of this architecture lie in the control of search.

The O-PLAN architecture has been applied in many domains and remains an active research platform at the University of Edinburgh Artificial Intelligence Applications Institute and has generated many papers [85].

4. Planning Under Uncertainty – the 1990s

a. SNLP

Systematic non-linear planners (SNLP) appeared in 1991 [86, 87]. SNLP is, by definition, non-linear, and uses lifting (allowing actions with variable expressions) as part of a least commitment strategy. An SNLP planner is also systematic, meaning that no plan or partial plan is ever examined more than once.

b. Pedestal

McDermott presented the PEDESTAL system in 1991, also [88]. Pedestal was the first (partial) implementation of ADL [31].

c. UCPOP

Penberthy and Weld developed a "Partial Order Planner whose step descriptions include Conditional effects and Universal quantification" in 1992 [89]. UCPOP (the name is an anagram of the capital letters in the name) operates with actions that have conditional effects, universally quantified preconditions and effects, and universally quantified goals. The planner uses "threats" to preconditions to trigger a resolution, either by reordering steps in the plan, posting additional sub goals, or adding new constraints. UCPOP uses the ADL representation. The authors can develop a completeness theorem for the planner. They prove that UCPOP is sound and complete and give several examples of how it solves problems described in earlier literature.

d. SENSp

Etzioni, et al., developed the SENSp planning algorithm in 1992 [90]. This algorithm is based on the UCPOP system and extends the work done on the SNLP planner to allow generation of correct plans in the presence of incomplete information. The SENSp planner operates on UWL, an extension to the STRIPS language designed to facilitate planning with incomplete information. The UWL extensions include annotations to preconditions and post conditions, the use of run-time variables, and extended truth values. The result is a provably correct algorithm for planning without complete information.

e. CNLP

Peot and Smith presented the Conditional Nonlinear Planning (CNLP) approach in 1992 [68]. CNLP extends the SNLP approach to allow for conditional planning. STRIPS operators are used, but extended to become *conditional actions* that may have several different mutually exclusive sets of outcomes. Although an interesting approach, CNLP suffers from a rapid expansion of complexity as more actions are added.

f. PRODIGY

Carbonell, et al., worked on the PRODIGY system, first reported in 1992 [91, 92]. PRODIGY explored two advantages of interleaving execution with planning: reducing overall planning and execution time and incorporating information from the environment into the planner's knowledge of the world. Stone and Veloso extended the PRODIGY algorithm to include prompts from the user and information that results from the execution of the user's direction [93].

g. RESUN

Carver and Lesser used partial hierarchical planning in the RESUN system, presented in 1993 [94]. RESUN interleaves planning and execution, and uses scripts and dynamic information gathering to refocus the problem-solver. Plan refinements are controlled by plan-specific heuristics, and the system can dynamically shift the focus of its attention.

h. Cost-Effective Sensing

Hansen presented an approach in 1994 that includes the cost of sensing into the determination of which actions to take in the plan [95]. He acknowledges that sensing after every action is acceptable so long as there is no cost associated with the sensing. If there is a cost associated with sensing it is reasonable to assume that sensing at intervals would be more cost-effective. A simple approach is to sense at constant intervals. Hansen notes that in many realistic environments some actions have different associated risks and/or error prone-ness associated with them. In such environments a variable-interval may be more appropriate. He presents "a generalization of Markov decision theory and dynamic programming in which sensing costs can be included in order to plan cost-effective strategies for sensing during plan execution." Although not directly applicable to military planning in a complex, adversarial, real-time environment, this work does suggest the utility of including an intelligent sensing strategy into a combined execution/planning system.

i. ZENO

Penberthy and Weld presented the ZENO planner in 1994 to handle actions that occur over long periods of time [96]. Deadlines for action commencement or conclusion are accounted for, and simultaneous actions that don't interfere with each other are allowed. ZENO is able to handle situations involving continuous change.

j. XII

Golden, et al., reported on the XII planner in 1994 [77]. This planner is based on the UCPOP algorithm, but interleaves planning and execution in the same fashion as IPEM. XII does not rely on the closed world assumption. Rather, it introduces the local closed world information (LCW) concept, which allows the planner to solve universally quantified goals in the presence of incomplete information. To do this, an assumption must be made that information that is available is correct. The action language is strongly related to ADL and to UWL

k. DRIPS

Haddawy and Suwandi implemented the decision-theoretic refinement planning system (DRIPS) in 1994 to reason with a probabilistic temporal world model [97]. DRIPS tries to maximize expected utility in terms of deadline and maintenance goals and the consumption of resources. Basically, the idea is to develop abstract plans, determine their associated utility, and prune away the known sub-optimal plans. This has the effect of focusing the planning effort onto plans that have a higher expectation of success.

l. Interval Reduction Strategy

Cohen, et al., discuss a technique for determining an efficient monitoring interval they call the Interval Reduction Strategy [98]. They present data on how well a monitoring policy based on this strategy performed in monitoring "Cupcake Problems." The Cupcake problem, due to a child development study by Ceci and Bronfenbrenner [99], involves a deadline or goal that must be met and determination of when the agent (software, human, bumblebee, etc.) should monitor the state to come as close to the deadline or goal as possible without overstepping it. The Interval Reduction Strategy can help determine an efficient monitoring schedule (in one- and two-dimensional Cupcake Problems) but relies on the agent being able to sense the entire state at each monitoring attempt and makes no allowance for the cost of monitoring.

m. BURIDAN

Kushmerick, et al., developed the BURIDAN probabilistic planner in 1994 [100]. Jean Buridan was a French philosopher credited with originating probability theory. This system uses a probability distribution over possible world states to model imperfect information. Actions are also modeled with probability distributions over changes to the world. Rather than attempt to arrive at a provably correct solution, BURIDAN builds a plan that is sufficiently likely to succeed, based on a user-specified threshold. The authors discuss a search control mechanism involving monitoring of the plan execution that can identify the point at which a probability of success drops too low. They state that the planner could use this information for more refinement of the plan. The authors also note that more work needs to be done in this area.

C-BURIDAN is a contingency planning version of BURIDAN developed by Draper, et al., in 1994 [101, 102]. The planning representation and algorithm are extended to include information-producing actions and the ability to exploit this new information. C-BURIDAN combines the new ability to model imperfect sensors with a framework for contingent action based on the CNLP algorithm. One interesting mechanism in the system is the idea of "branches" that connect information-producing actions to subsequent actions that require that information. The resulting system can build plans in which different actions are executed depending on the outcome of previous actions.

n. PLINTH

Goldman and Boddy presented the PLINTH conditional linear planner in 1994 [103, 104]. This system is based on McDermott's PEDESTAL system and treats contingency plans much the same way that CNLP does. [64]. PLINTH accommodates conditional actions, whose effects cannot be predicted with certainty. Noting that conditional linear planning is simpler than conditional non-linear planning, the authors applied PLINTH to planning image processing actions for NASA's Earth Observing System.

o. Dynamic Programming Envelopes

St. Amant, et al., describe the idea of "envelopes" in dynamic programming to monitor the progress of an agent in accomplishing its goal [105]. Essentially, the envelope represents the portion of the state space in which the goal can still be reached. When the agent moves across the boundary of the envelope it is certain to fail in

achieving its goal. The important idea here is that the agent ought to be aware of the envelope and avoid the boundaries. This approach assumes that the agent is monitoring its own progress towards accomplishing the goal and constructs its own plans. The idea of watching progress, being concerned with how close the actual state is to known failure states, and initiating re-planning is central to the Execution Monitor and Planner used in this research.

p. COLLAGE

Lansky examined an approach to domain representation and planning based strictly on actions and their interrelationships, rather than on state-based goals and preconditions [106]. This "action-based planning" approach was implemented in the COLLAGE system. COLLAGE is essentially a constraint-satisfaction planner, but the constraints are on actions that can be taken.

q. Active Decision Postponement

Joslin and Pollack examined the effects of considering postponed decisions in current decisions in 1995 [107]. They note that planning systems that postpone decisions and don't consider them in current planning (which they call *passive postponement*) often make incorrect decisions causing simple tasks to become intractable. They propose *active postponement*, a technique that includes constraints from postponed decisions in current reasoning about the plan. This technique can break the problem into sub-problems that are easily solvable by standard constraint satisfaction methods. They caution, however, that there are many problems where an early-commitment strategy yields a more efficient solution.

r. Sage

Knoblock developed the Sage system in 1995 to address several problems arising in gathering information from large networks of distributed information [78]. These problems include replicated information, parallel execution of actions, failure of actions due to problems with remote resources, and the need to interleave sensing with execution. The Sage planner builds extends the UCPOP algorithm to support simultaneous action execution and to integrate planning and execution. When an action fails, Sage re-plans to remove the failed portion of the plan and work around it.

s. <I-N-OVA>

Tate presents the Issues - Nodes – Orderings/Variables/Auxiliary (<I-N-OVA>) approach to representing and manipulating plans[108]. This approach was intended to assist in connecting different work on formal planning theories, practical planning systems, and process management methodologies, and is based on representing plans as a set of constraints. The constraints are of three general types: issues, nodes, and detailed constraints. The detailed constraints are ordering constraints (temporal or metric), variable constraints, or auxiliary constraints (point in the plan, or range across the plan).

t. Cassandra

Pryor and Collins presented details on the Cassandra contingency planning system in 1996 [64, 109]. Cassandra is a SNLP partial-order planner able to develop plans that allow for uncertainty. Modified STRIPS operators represent actions, and each possible effect has an associated set of secondary preconditions that define the

conditions that will cause that particular operator to be selected. Cassandra makes a distinction between decision steps and information gathering steps, and distinguishes between the possibility of performing and action and the necessity of performing it. Cassandra exhibits the same problems of exhaustive search and requires effective search heuristics to keep even simple problems from becoming impractical to solve.

G. Military Modeling, Simulation, and Planning

Planning under uncertainty becomes even more complex when there is an adversary actively trying to defeat the plan. The complexity increases even more in the military planning domain, where the outcomes of actions are very probabilistic. Some approaches to dealing with this complexity have been proposed and some systems have been developed to deal with portions of the problem. Artificial Intelligence techniques and simulation are being exploited to tackle these sub-problems. This section provides a review of the approaches and the systems, again in roughly chronological order.

1. Automation of Command and Control Processes

Partridge stated the need for automated support throughout the spectrum of military operations, and proposed four distinct modules to help the human decision-makers [110]. These modules are a Mission, Enemy, Terrain, Time, and Troops (METT-T) evaluator, a course of action optimizer, a rehearsal support tool, and a rapid decision-maker. The decision-maker would use the course of action produced by the war-gaming process as a baseline, accept situation updates as the battle progresses, and use a genetic algorithm optimizer to recommend decisions.

Kelly, et al., describe a prototype that implements a Command and Control Decision-Support Architecture developed in a layered distributed object-based environment [111]. The lowest layer is the Data Model that maintains the objects representing the course of action planning information. The next layer up is the Controllers layer that handles object creation and the command and control war-gaming engine. At the top is the Applications layer that provides the user interface into the system. The architecture uses a publish-and-subscribe methodology to ensure any modules that require information from the system can get it as needed. Kelly also describes how their prototype demonstrates support for four decision support concepts: COA Development, COA Analysis, Execution Monitoring, and COA visualization.

Seligman, et al., discuss an example of execution monitoring applied to solving the dilemma between overloading users with information and excluding too much information [112]. They describe a decision-centric information monitoring (DCIM) approach that identifies information that is critical to known decisions, places a higher priority on that information, and filters the available information to ensure these "critical information needs" rise to the top [113]. Their prototype system, called LOOKOUT, applied the DCIM model to the logistics domain and demonstrated performance gains in extraction of useful and timely information.

Wynn, et al., present a mechanism for supporting the COA visualization concept by simultaneously producing a three-dimensional and two-dimensional visualization of a course of action under development for consideration by the staff [114]. Such visualization allows all of the planners to retain a common understanding of the COA.

The Army Modeling and Simulation Office (AMSO) has identified technology voids in the areas of automated decision aids, COA tools, and tactical information aids [115]. This project could support all three of the areas mentioned.

The Army Research Laboratory (ARL) is generally focusing on developing the infrastructure to support command and control decision-making (visualization, software agents, collaboration tools, multi-modal interaction, etc.) [116]. ARL is also funding a research program in Intelligent Information Processing for Visualization [117].

Kirzl examines how the rapid acceleration of information exchange on the battlefield will impact command and control processes and increase the speed and quality of decision making [118]. He identifies "adaptive decision making" based on information systems automating simple and compound/contingency decisions, leaving the decision maker and staff to focus on complex decisions. He also envisions information systems facilitating "merged planning and execution processes." He extends his analysis to include "measures of merit" for the assessment of future command and control. One of these measures of merit is "more explicit uncertainty management," in which information is flagged as incomplete or is provided with a confidence tag related to the ground truth. Several of the measures of merit relate to adaptive decision making, and include the capability of decision support tools to generate and assess alternative futures and courses of action, a rapid plan/re-plan capability, and contingency rich course of action analyses and plans.

Tolk identifies the requirements for simulation systems used as part of a decision support system [119]. Among these requirements are that all command and control

processes must be adequately modeled, command agents and computer generated forces have to be used, the initial state of the simulation must be generated from actual data from the command and control system, and adequate and validated data must be available for the simulation system.

The Defense Modeling and Simulation Office describes its vision for using modeling and simulation in support of planning in the Defense Modeling and Simulation Master Plan [120]. It states that "M&S will be used to assist in the development and evaluation of operational plans at all levels. Significantly, "decision-makers will be able to simulate and evaluate the consequences of alternative courses of action during deliberate and crisis action planning."

The Defense Advanced Research Projects Agency completed a proof of principle pilot test on a Course of Action Analysis (COAA) system [121]. This system examines one of the initial steps in provided integrated support for continuous planning and execution of military operations. The focus of the COAA project is studying techniques for improving the COA analysis step and on aiding the decision-makers in understanding of alternative COAs. The current scope of the project ends with COA comparison, but future work in plan generation and monitoring could extend the scope to the entire spectrum of military operations. One of the lessons learned in the COAA project was that COAA tools must be tightly integrated with the planning process rather than being stand-alone tools. Another lesson learned was that a common plan representation is critical to integrate automated planners and decision support tools. Also, the COAA project highlighted how efficient analysis tools can fundamentally change the planning

process. One of the open research issues identified by this project is the addition of a full set of doctrinally correct military tasks to make the system more robust.

Alberts discusses the future of Command and Control in an environment where U. S. forces have "Dominant Battlespace Knowledge" (DBK) [122]. He points out that DBK can yield "option dominance" in which friendly forces can generate options and respond faster than the enemy forces. Alberts identifies some of the prerequisites for option dominance as understanding the current situation, the generation of options to be considered, analysis of those options, and a command decision to select an option. Of course, the key point is that U. S. forces must be able to accomplish all of these prerequisites faster than the enemy can react. He also notes the requirement for "a more streamlined process ... to satisfy the time-critical nature of this task." One particularly insightful point is that if U. S. forces can demonstrate option dominance to potential adversaries it may be possible to preempt enemy actions and prevent combat.

Brandt explores some of the issues involved in linking modeling and simulation with command and control systems [123]. He notes how establishing well-designed links between the two can enhance course of action development, analysis, and selection, as well as support the rehearsal process.

2. Military Simulation

Kang provides a good review of military simulation [124], the details of which will not be pursued here. However, it is appropriate to consider several issues in military simulation. Simulations in the U. S. military can be used for analyzing strategy, operations, and tactics, but are primarily used for training. For example, the Theater-

Level Campaign Modeler [9] and the Institute for Defense Analysis Tactical Warfighter (IDA TACWAR) [19] are strategic level simulations. The MODSAF system is an example of a tactical level simulation [125].

Training simulations are used to train the gamut of individual level skills through unit level activities. For example, individual skill trainers include devices used to train individual marksmanship before using live ammuniction. Others are used to train crews, such as helicopter simulators and Unit-Conduct of Fire Trainers (U-COFTs). At the unit level, Janus (individual vehicle level through company or battalion) [126], BBS (Brigade-level and below), and CBS (Corps-level simulation) are used to aid in training. However, in a classic example of the inappropriate use of simulations mentioned earlier, the military is applying these last three systems as training aids when they were really designed for analysis. Several new initiatives, such as WARSIM, are underway [127].

One of the significant problems the military faces is that training simulations require large facilities and a great number of personnel (contractors and soldiers) to run them. Another problems is that so many simulation systems have been developed for (or applied to) training and they are all proprietary. The military is addressing this problem by using a Distributed Interactive Simulation (DIS) protocol to allow these different systems to work together [128], but this is a "patch" not a solution.

Since the military is such a large organization, simulations must be targeted at the appropriate level. For example, it would be inappropriate to produce thousands of Janus systems and link them together to form a Corps level simulation. It is more

appropriate to aggregate lower levels. Hoeber presents a hierarchy of military simulations based on the level of aggregation [19].

Identifying the problems with current military simulation is simple, particularly when none of them fit neatly into the support of ongoing operations. Surdu identifies the desired capabilities for a military on-line simulation [129], most of which were supported by the Blais' work on the MEWS system [130]:

- The simulation must be executable from a single workstation by a single user.
- The simulation must be executable on low-cost, open-system, multi-platform environments.
- The simulation must be capable of running in multiples of wall-clock time (i.e., real time and much faster than real time).
- The simulation must be able to receive and answer queries from external agents.
- If needed, multiple simulations should be capable of operating together.
- The simulation should be based on an aggregate-level model.
- The simulation should interface directly with military command and control systems [7]

3. Simulation Support for Military Planning

Lee and Fishwick proposed integrating simulation into the planning process as a new way to perform intelligent reactive planning [131]. A common sequence is apparent in many strategies for handling the complexity of reasoning in reactive

planning. First, candidate plans are generated, and second, they are evaluated. In 'Simulation-Based Planning' simulations are used instead of rules to evaluate generated plans. Simulation-based planning has application in the military planning arena because it lends itself to adversarial and multi-agent planning.

Lee's dissertation describes how Simulation-Based Planning extends planning [132]. Simulations, rather than analytical solutions, are used to handle probabilistic uncertainty. The simulations necessarily enable a higher level of detail, resulting in plans that are much more closely related to the actual execution. Lee and Fishwick further describe the embedding of simulation to resolve the actions of the entities before committing to a plan [133]. Entities are individually simulated so that all of their possible responses to the proposed plan can be considered.

Anderson segregates simulation from the planner as an approach to dealing with real-time planning situations [134]. He developed the Multiple Event Stream Simulator (MESS) to provide a domain-independent simulation system that could be queried by a separate planner. Although the focus of MESS is on the simulation piece, Anderson identifies how useful it is to have the planner act as a separate agent that can monitor the execution of the plan, scrap failed portions of the plan, and use simulation to conduct re-planning.

Surdu, Haines, and Pooch describe the requirements for operationally focused simulations [129]. The simulation must be able to run on a single workstation with a single operator. It must run on low-cost, open systems, multi-platform environments. It

must be capable of running in multiples of wall-clock time. Finally, it must be able to receive and answer queries from external agents.

The benefits that can be gained from using simulation to support planning have been identified by Fishwick, Kim, and Lee., [135]. They describe the goal of allowing simulation to be used in real-time, where the simulation is embedded within the decision-making system. The primary advantages of such a system are the ability to run in much faster than real-time, a general utility for obtaining answers to "what if" scenarios, and the ability to tailor the detail and execution.

Blais and Garrabrants describe the Marine Air-Ground Task Force Tactical Warfare Simulation (MTWS) and demonstrate how it fits the commander's need for a planning and rehearsal system to support operational planning [130].

Barone and Roberts discuss potential uses for simulation in military planning [136]. Their SimLink project connected the Battlefield Planning and Visualization (BPV) concept demonstrator with the Eagle combat simulation system. One of their suggestions is that a simulation could be run in parallel with the actual operation to provide an automated execution monitoring system that would identify divergence from the plan and initiate re-planning. A key contribution of this paper is their discussion of a "C2 Schema" which forms a master representation of the plan that can be translated into plan representations for both the BPV demonstrator and the Eagle Simulator.

Sheehan, et al., describe the Order of Battle (OB) Data Interchange Format (DIF) that the Defense Modeling and Simulation Office (DMSO) has proposed to provide a

consistent and easily replicated representation of forces between simulations [137]. OB data will be one of several components in the Plan Description.

The director of the Army Modeling and Simulation Office (AMSO) described the difference between using simulations to examine COAs after they were manually produced and a large-scale simulation capable of developing its own courses of action [138]. There are difficulties related to the size of the solution space, and using simulations as a "solution solver" to determine effective COAs remains a major technological challenge. Although small-scale efforts have been produced, there is no major system under development. He identifies a twenty-year gap before such major systems are in place.

4. Software Agents in Adversarial and Military Planning

Software agents are notoriously difficult to define, since the title can be applied in many ways. Russell and Norvig define an *agent* as "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors" [31]. Franklin and Graesser provide a taxonomy of agent types, of which software agents are one branch, and a description of agent properties. Among these properties are reactivity, autonomy, goal-orientation and temporal continuity [139].

Lejter and Dean identify several agent control strategies [140]. Among these is the "request-response" strategy in which the agents are organized into a hierarchy. The higher up agents handle the most complex tasks, and break them up into subtasks for execution by agents in the next level down.

Spector and Hendler describe the supervenient Agent Hierarchy for integrating planning and reaction in complex, dynamic environments [141]. In summary, supervenience transmits goals down the agent hierarchy and requires lower-level agents to report sensor acquisitions and other information up the hierarchy.

Bouche, et al., describe the use of a "command agent" to simulate the decision-making processes at various command levels in an operational simulation [142]. The agents, which represent commanders at different echelons, develop Courses of Action (COAs) that are then run through a simulation. The results are used to refine the COAs.

5. Adversarial / Military Planning Systems

a. PHOENIX

Cohen, et al., presented the PHOENIX system in 1989 [143], and later expanded it into an adaptable planner for a complex, real-time environment. PHOENIX uses a least-commitment strategy they call "lazy skeletal refinement" and a combination of reactive and deliberative planning components. Another key piece is a monitoring construct that gives advance warning when a plan is failing [144]. This advance notice is useful in adapting the plan as it executes [145]

b. Applegate's Architecture

Applegate, et al., examined the additional complexity that AI planning techniques face in adversarial situations [146]. Among these problems are that the environment is unpredictable and dynamic, that the plan must be adjusted dynamically during execution of the plan, and that the presence of an adversary must be considered during plan development. This means that the AI planner must make assumptions about

the outcome of events, and re-plan as necessary. These assumptions can then be monitored during the execution of the plan.

Applegate also discusses an approach to Plan Representation that places "less significance on states of the world than on the derivation and persistence of desired conditions in the world. Leaves in the plan representation represent orders to units rather than discrete executable actions, but still require many of the aspects of traditional AI planners (list of actions, temporal constraints, variable bindings, and preconditions). Applegate's approach assumes agents that represent units, which is different than the approach taken by this research, but many of her points are still relevant. For instance, intelligence gathering gains added importance, since knowledge of the actual situation will trigger re-planning.

Applegate's scheme handles simultaneous execution of actions by maintaining a "play-list" in which all simultaneous activities are concurrently considered. Active plays are those with a start time prior than the current time and an end time later than the current time. These plays are associated with the actions that will be taken by the unit agents, and planned plays are kept in the play list until they are invalidated.

c. CYPRESS

Wilkins and Myers in 1994 describe the CYPRESS system that provides the framework for the creation and control of taskable, reactive agents [147, 148]. Taskable, reactive agents have two main components: an executor and a planner. The executor constantly monitors the world state for situations requiring it to take action. The planner synthesizes sequences of actions that serve as a template for later refinement by the

executor. Communication and coordination between the executor and the planner are accomplished through the ACT formalism [148]

d. Trajectory Management

Gilmer and Sullivan, beginning in 1996, discuss their work on management of multiple outcomes resulting from an event in a stochastic simulation [149]. Rather than allowing trajectories along every possible outcome, they restrict the possible outcomes to a set of representative outcomes with associated probabilities. Their more recent work includes assessment of different implementations, including a discrete event simulation approach, a tail-recursive approach, and a state duplication approach [150]. They have reported positive results in converging towards the set of representative outcomes, but caution that there are still some limitations to the approach [151]. Al-Hassan has investigated the use of measures of effectiveness to prevent the problem of discarding interesting outcomes that have low probabilities [152]. The modified system is designed to be sensitive to loss ratios while determining representative outcomes. Their most recent work has focused on recursive simulations, wherein the simulation entities themselves invoke instances of the simulation to explore the outcome of decisions [153].

e. Adversarial Planner

In 1997, Elsaesser described an Adversarial Planner (AP) which addresses the complexity of battle planning by limited the problem space through determination of the adversary's counterplans, monitoring of execution, and replanning when the original plan is in jeopardy of failure [154, 155]. AP uses task decomposition planning to develop a

complete plan in layers, and defers expansion of the plan until it is required. Counter-planning is used to represent possible adversary plans and determine ways to defeat them.

f. FOX-GA

FOX-GA, developed by Hayes and Schlabach in 1998, is a tool that uses course-grained representations in order to provide timely COA generation and assessment [156]. Its relation to this work lies in its use of a genetic algorithm for allocation of assets, but at the higher brigade COA level [157]. FOX-GA will be transitioned to the Communications-Electronics Command (CECOM) to be part of the Command Post XXI Advanced Technology Demonstration [158, 159].

g. FGDO

Army Major Robert H. Kewley, Jr., combines fuzzy inference systems with genetic algorithms in 1999 to form a fuzzy-genetic decision optimization (FGDO) system that he applied to the battalion-level tactical course of action (COA) development problem [160]. In his system a fairly sophisticated tactical simulation module is used to evaluate the outcome of proposed COAs. The performance of each COA is fed into a fuzzy preference module. From this module an overall fitness for the COA is fed back into a genetic algorithm module that continues to produce modified COAs. Kewley's approach differs from this project in that he focuses at the higher (battalion) level course of action and uses a sophisticated simulation. Naturally, it takes much longer to solve such a complex problem. This project, although it could be used at battalion level, is focused more at the individual tank or platoon level and uses a simple combat results

mechanism. The similarity between the two projects lies in their use of genetic algorithms to determine better outcomes. Also, Kewley's project recommends future work on biasing the initial selections, which is a fundamental part of this project.

h. GRASP

Atkin, et al., starting in 1998, describe the use of a multi-goal partial hierarchical planning approach to planning in continuous, uncertain, adversarial real-time domains [161]. They developed the General Reasoning using Abstract Physics (GRASP) planner to address four problems encountered by planners in such domains: resource allocation among multiple goals, determining plan operator effects, reacting to and exploiting unforeseen events, and generating workable plans quickly.

The GRASP planner uses a combination of several techniques to address these problems. First, plans are not generated from atomic planning operators at run-time. Rather, a general solution is developed and expanded. Second, a simulator is used to establish the world state after a plan has executed. Third, state boundaries are created dynamically as plans are executed using critical points to mark the boundaries. These critical points are generated by simulating forward to determine the post-conditions of execution of the plan to that point. Fourth, the planner operates at a fairly high level and relies on plan operators to be competent and cope with unforeseen events.

Atkins's GRASP planner extends the partial hierarchical planning framework by explicitly representing multiple goals and integrating the planner into an action hierarchy, Hierarchical Agent Control (HAC), that handles resource arbitration and failure recovery [162, 163]. HAC provides a general skeleton for controlling agents and

for management of sensing information, scheduling of actions, and message passing. The action taken at each level, following Spector's supervenient concept, is that messages from lower-level actions are processed, the state is updated, new lower-level actions are scheduled, and any required messages are sent up to the parent. HAC manages resources by leaving the allocation of sources between lower-level actions up to the higher-level action. In HAC, a forward simulation process evaluates plans that denote an action that satisfies a goal. In this way, the planner determines what would happen if the plan were executed. Atkin, et al., have implemented their approach in the Capture the Flag development domain [164]

i. MEWS

The modified version of ModSAF (called MEWS) presented by Porto, et al., in 1999 focuses on platoon-level course of action generation in an environment where two competing platoons must encounter each other on the way to their objectives [165]. Different goal parameters can be set, such as the importance of timely arrival at the objective, the importance of survival, or the importance of eliminating enemy tanks. An adaptive algorithm drives the behavior of one or both sides in the conflict, and the evolutionary algorithm compares possible tactics based on the success parameters.

j. OpSim

Surdu, Haines, and Pooch [129, 166] developed a system called OpSim in 1999 designed to monitor the current operation. The result of that research verified the feasibility of their implementation of Execution Monitors that use simulation to determine the significance of differences between the execution of the operation and the

plan. OpSim uses a dynamic hierarchy of rational agents, called Operations Monitors to compare the current situation with the plan. The top-level Operations Monitor informs the decision maker when the success of the plan is at risk.

k. SIPE-2

Wilkins and Desimone applied the SIPE-2 planner to the military domain by building the System for Operations Crisis Action Planning (SOCAP) [167]. Within the sub-domain of military transportation, this system successfully generated employment and deployment plans for getting combat and support forces to the desired locations at the right time.

l. Tactical Event Resolution

Hill and Miller successfully combined software agents, crisp reasoning, and a genetic algorithm to resolve tactical events [168]. This 1999 work verified the applicability of genetic algorithms to generation of options in a course of action with a niching strategy based on battlefield function biases as a heuristic to restrict the initial populations to those with a reasonable expectation of success.

m. DARPA SUO/SAS

Tate, et al., reported on their application of the O-PLAN architecture to Army small unit operations in 2000 [169]. The system they developed is called the Defense Advance Research Projects Agency (DARPA) Small Unit Operations (SUO) Situation Awareness System (SAS). SUO/SAS demonstrates how artificial intelligence planning techniques can be useful in building a planning and decision aid for small units operating in urban terrain.

CHAPTER III

DESIGN

A. Methodology

In this chapter the methodology for an Anticipatory Planning Support System (APSS) is presented and the design of the system described. See Figure 1 for a depiction of the methodology. The methodology has been implemented in an *APSS* prototype to enable evaluation of the methodology and its subordinate processes. For clarity, components of the methodology are capitalized and italicized.

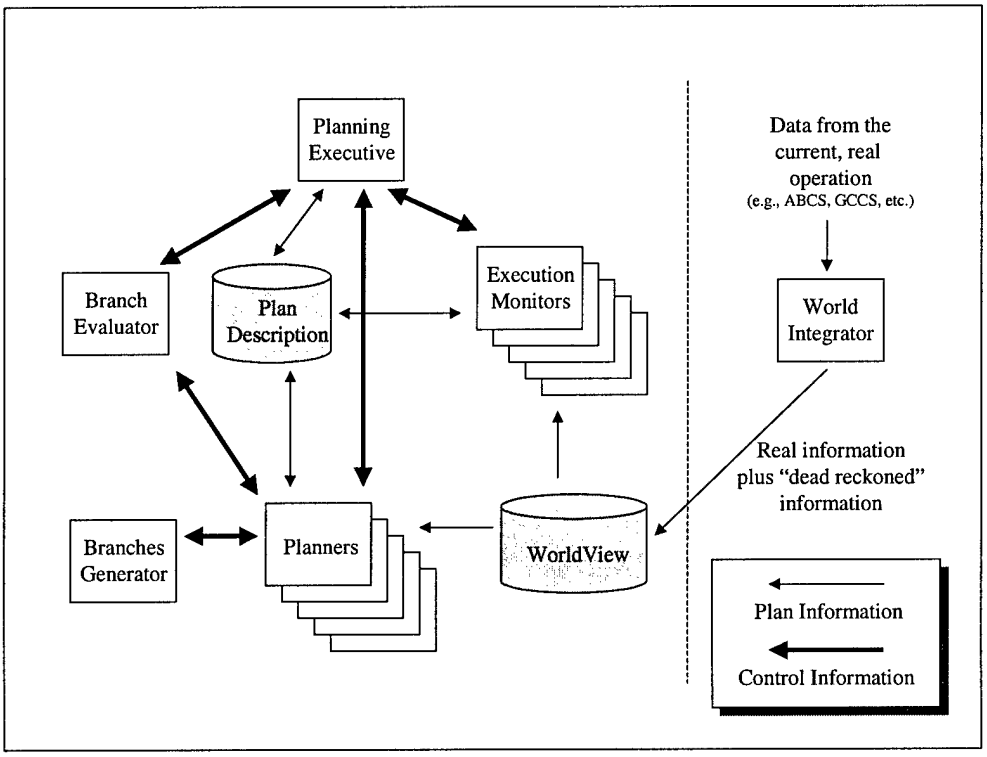

Figure 1: Anticipatory Planning Support System Methodology

Information collected during a military operation is processed through a *World Integrator* to generate a *World View* that provides the *Actual State* of execution. A *Planning Executive* controls the anticipatory planning process and the use of system resources. A *Plan Description* represents the plan tree and manages modifications to it. *Execution Monitors* compare the *Anticipated State* of the plan at a particular *Node* with the *Actual State* of the execution and notify the *Planning Executive* if there is a potential problem.

The *Planning Executive* launches *Planners* to generate and evaluate new *Branches*. A *Branch Generator* uses a genetic algorithm combined with inference mechanisms to produce new *Branches*. A *Branch Evaluator* examines a *Branch* to provide *Planners* and the *Planning Executive* with viability measures and outcome confidences. The *Execution Monitors* and *Branch Evaluators* use simulations to perform their evaluations.

The human planners will not accept or rely on the system unless they understand the system's "logic." If the recommendations of the system "make sense" to the human planners, or if the system provides a reasonable explanation capability, then it is more likely to be accepted and used. Regardless of how flexible and sophisticated the simulation and analysis system is, it still may not provide results that the planner will accept. Accordingly, the system provides the means for the human planner to override the results with an outcome that makes more sense. This postpones the need to re-code the event resolution mechanism or the simulation.

B. Capturing and Representing the Actual Situation

The methodology requires a representation of the *Actual State* of the operation. Surdu and Pooch describe the use of a *World Integrator* and *World View* system to provide the *Actual State* [166, 170-172]. The *World Integrator* and *World View* involve issues in sensor, data, and information fusion. The *World Integrator* must determine when an entity has been unconfirmed long enough that its actions must be dead reckoned. When some sensor reports a similar unit, the *World Integrator* must determine whether this is merely the lost unit reappearing or a different unit. These and other issues regarding sensor, data, and information fusion are open research issues, and are not implemented in this prototype. Rather, a synchronized simulation is used to provide specific *Actual States* to the *Anticipatory Planning Support System*, stimulating the prototype for evaluation purposes.

1. World Integrator

The *World Integrator* has the onerous task of monitoring the real operation, processing that information, and passing it to *World View*. In some systems, such as the Global Command and Control System (GCCS), this may involve querying a database [173]. In other systems, this may require "eavesdropping" on the network. The reason for this intermediate step is that in real operations, reports on some entities may be intermittent. It is the job of the *World Integrator* to "dead reckon" these intermittent reports and pass them into *World View*.

2. World View

The *World View* module is a representation of the real operation. In order to make the job of the *Execution Monitors* easier, the representation of the real operation and the *Plan Description* should be as similar as possible. *World View* receives information about the state of the real operation through a series of APIs. It then transforms this information into a form that the Execution Monitors can easily interpret. Clearly, when an entity has been "dead reckoned," this must be reflected in the information that *World View* presents as the *Actual State*.

3. Actual State

In a real military operation, the *Actual State* of the operation would be provided by real command and control assets, such as the Maneuver Control System (MCS) [174] funneling information through the *World Integrator* into the *World View*. For the purposes of this research, the *Actual State* is produced by an external mechanism that represents the activities of the *World Integrator* and *World View* components. A separate *Plan Description* with controlled differences from the *Plan Description* built by the human planner is processed by the external mechanism using a discrete event simulation to produce the *Actual State*. The *Plan Description* and the simulation in the external mechanism must remain synchronized with the *Plan Description* and simulations used by the prototype *APSS*.

C. Representing Entities

When modeling the domain in which planning occurs, two fundamental things must be determined. First, the modeler must identify and define all of the participating entities. Second, the modeler must determine all possible interactions between the entities and the effect of those interactions on the state of the system. Although the anticipatory planning process should be applicable in any planning domain, the focus of this research has been limited to the military planning domain. Accordingly, the entities of interest are tactical entities and the terrain upon which they operate.

1. Tactical Entities

Tactical entities can be defined in many ways, including their affiliation (enemy, friendly, etc.) and their category (unit, obstacle, artillery burst, etc.). Within their category, they have several other distinctions. For instance, units have roles (armor, mechanized infantry, etc.) and levels (platoon, company, etc.), while obstacles have types (area, linear).

Tactical entities also have different capabilities. Units can move or remain idle, and participate in an engagement. Obstacles don't move, but do have build-up times before they are effective, and have varying effects on other entities. Artillery bursts and obstacles have a limited effective range, whereas units have much larger effective ranges.

Tactical entities also have many attributes that can vary during the course of an operation, and therefore can vary in the plan. Among these are the strength of the unit

and the amount of resources on hand. For the prototype system, only a limited set of these attributes, such as strength, fuel, and ammo, is tracked.

For the *APSS* prototype several types of units, one type of obstacle, and one type of artillery burst are implemented. Although the prototype implements tactical entities ranging from individuals through brigades, in practice only company, platoon, and section sized units are used. This is necessary to enable the system to have enough flexibility to develop realistic, descriptive, plans.

2. Terrain

In military operations the terrain plays a significant role in how units move and interact. Terrain has many attributes, including trafficability of the surface, amount of vegetation, and how built-up it is with buildings. Also, hydrology (rivers, lakes, streams, etc.) is a consideration. Also, elevation is a factor in whether units have line of sight on other entities. Sometimes, the effects of the terrain can be mitigated by road systems.

Although terrain is quite variable in all of its attributes, it is sufficient for modeling purposes to define a small region of the terrain and assume that the attributes are consistent or representative within that region. The size of the region is dependent on the resolution required by the modeler.

For the *APSS* prototype a region size of approximately one square kilometer is used. This is appropriate when the largest unit being used is a company. On the terrain, a company typically occupies an area of about one square kilometer. The selection of kilometer-square regions also provides another benefit in that it allows for discrete changes in the locations of units.

Although unit movement in the military domain is continuous, it is sufficient for the prototype to merely track changes in location of one kilometer or greater. The discrete movement of units is particularly useful since the prototype employs discrete event simulations.

3. Interactions

The interaction of the entities with each other is what causes changes in the state of the system. The primary interaction between tactical entities and other tactical entities is engagements. During engagements, which occur over a period of time, the opponents can lose strength (weapon systems) and consume resources (ammo). The primary interaction between tactical entities and terrain is movement. The attributes of each region of terrain have effects on tactical entities, primarily in terms of how much time it takes to traverse the region and how much fuel is consumed. There is no interaction between terrain entities, although border conditions must agree where terrain entities are adjacent.

D. Representing the Plan

The *Plan Description* is a representation of the possible ways the operation can proceed (see Figure 2 for a depiction). The *Plan Description* is a directed tree with the possible states of the plan held by *Nodes*. The *Branches* of the tree represent the changes between states caused by the sequence of actions of the friendly and enemy participants.

Note that the *Plan Description* is not a game tree for resolution of a minimax problem, in which each level represents a turn by the adversaries. Russell and Norvig describe the use of such a game tree and the minimax algorithm [31]. Instead, each *Branch* is the collection of multiple and concurrent actions of the participants. After the actions have been performed and the interactions resolved, the *Node* at the end of the *Branch* contains the resulting *Planned State*.

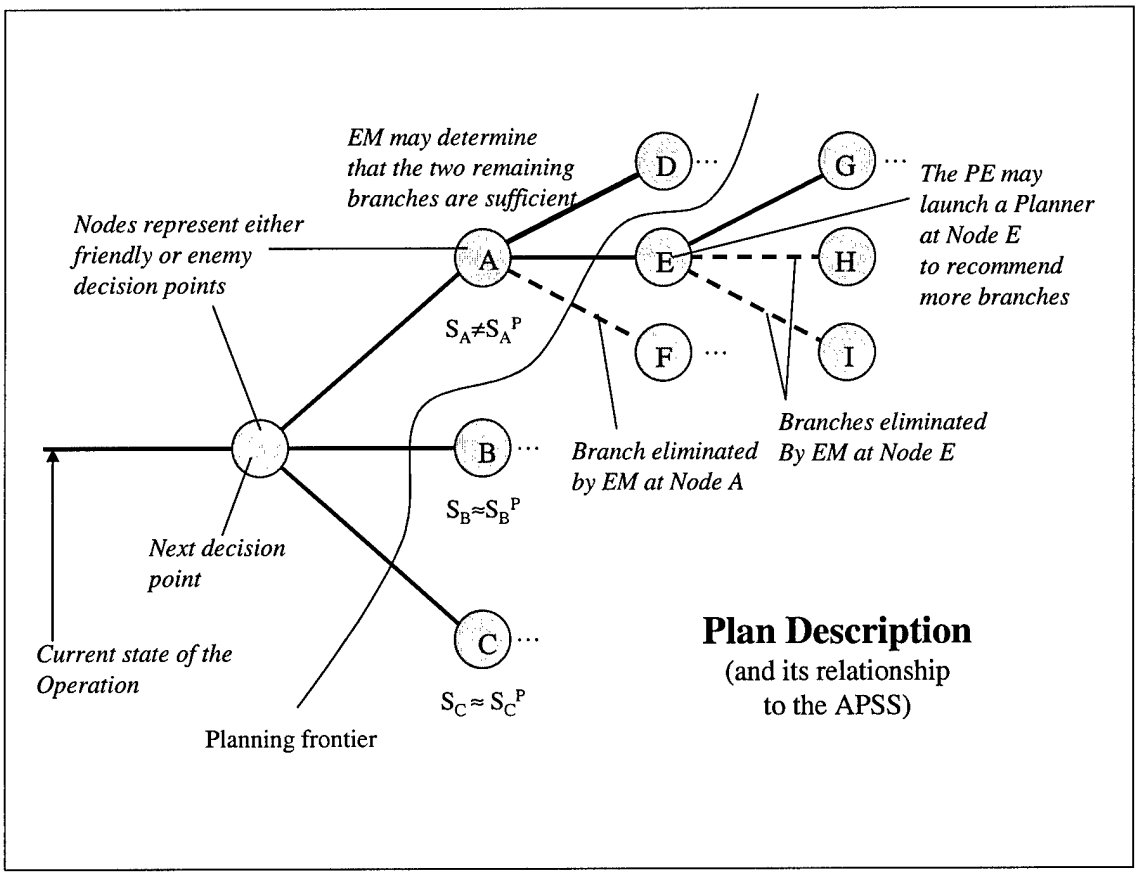

Figure 2: Depiction of a Plan Description

1. States

In most simulation systems, a state is the "minimal collection of information with which the system's future state can be uniquely predicted in the absence of chance events." [5] Although the simulations used in the *APSS* do concern themselves with every transition in system states, the *Anticipatory Planning* process does not attempt to track every specific state of the operation. Rather, it mimics the approach human planners use when they think about constructing a plan. That is, certain "critical states" are considered, either because they represent a significant conclusion of activities, or because they represent a place/time where the plan can diverge.

There are three kinds of states maintained in this system: the *Actual State*, the *Planned State*, and the *Anticipated State*. The *Actual State* comes from the *World View*. A *Planned State* is generated when a *Planner* initially creates a *Branch* in the plan, and is held in a newly-created *Node* in the *Plan Description*. If an *Execution Monitor* is observing a *Node*, it periodically creates an *Anticipated State* by using simulations to project the *Actual State* forward to the time of the *Node*.

2. Nodes

Each *Node* holds a *Planned State* that includes the state (location, strength, etc.) for each tactical entity. The *Nodes* connect to any *Branches* that have been produced by *Planners*. As the plan is constructed, particularly if valid changes are made early in the plan, the *Nodes* are responsible for propagating the changes to all following *Nodes*. The relative time stamp associated with each *Node* is dependent on the relative time stamp of the previous *Node* and the actions taken by the entities within the intervening *Branch*.

Thus, it represents the earliest time that the *Planned State* held by the *Node* can be achieved. The *Nodes* also provide an important function in communicating the viability measure associated with *Branches*. The viability of the *Node* at the end of a newly planned *Branch* is a weighted function, over all of the entities, of the ratio of their actual strength against the desired end strength and their distance from the objective. Measures of viability are computed for *Branches* after planning or re-planning and are propagated towards the trunk of the tree by the *Nodes*. Similarly, the *Nodes* propagate tactical entity state changes by adding or subtracting an offset amount for a particular attribute (fuel, ammo, strength, etc.).

3. Branches

A *Branch* represents action taken by the friendly and enemy forces that result in a new *Planned State*. The actions have associated preconditions, viability measures, and a confidence measure. This is similar to the action-based approach to planning Lansky presented in the COLLAGE system [106]. The difference lies in the way that COLLAGE uses unsatisfied constraints to direct the execution of the system, whereas *APSS* incorporates a priority scheme that the *Planning Executive* uses to control when and how much planning is done.

Within the constraints placed on the *Planner* by the *Planning Executive*, the best series of action choices that become *Branches* in the *Plan Description*. The *Planner* determines which candidate *Branches* are the 'best' by applying a fitness function that weights the friendly and enemy viability measures and then choosing the *Branches* with the best fitness.

The commander may desire to add a new *Branch* to the plan manually, typically at a place/time that the commander will cause the plan to diverge based on a decision point. The new divergence in the plan is represented in the *Plan Description* as a new *Branch* from whichever *Node* contains the *Planned State* where/when the divergence must occur. Then the commander can manually construct the *Branch*, or a *Planner* can be used to complete the *Branch*. Also, a *Branch Evaluator* can be used to assess the viability of the *Branch* for the commander.

E. Determining Outcomes with Simulations

A variety of simulations could be used to support the *APSS* prototype, ranging from high to low resolution. For instance, the level of resolution required for the *Planner* might be less than the level required for the *Execution Monitors*. Time or system resource constraints may dictate that *Planners* and *Execution Monitors* be able to select the simulation with the appropriate resolution to provide "good enough" answers "fast enough."

Surdu, Haines, and Pooch describe the requirements for such operationally focused simulations [129, 166, 170, 171]. They include the ability to run on a single workstation, on low-cost open systems, and in multiples of wall-clock time. Also, the simulation should be able to answer queries from other agents. Other requirements are that the simulation should be capable of working in cooperation with other simulations, and it should be based on an aggregate-level model. The simulations used in the *APSS* prototype satisfy all of these requirements.

Fishwick, et al., [135], reiterated by Blais and Garrabrants [130], have identified the benefits that can be gained from using simulation to support planning. Foremost among these is the support for the conduct of "what-if" analyses in much faster than real time. This is possible because modern simulation systems can represent a large number of effects and entities and run scenarios very quickly. The ultimate benefit, of course, is that commanders will be able to make better decisions sooner than possible without such simulation support.

1. Types and Capabilities of Simulations

This methodology does not rely on any particular simulations. Any simulation used to support the Anticipatory Planning process must be able to accept a state (*Actual State* from the *World View, Planned State* from the *Plan Description*, or *Anticipated State* developed by an *Execution Monitor*), treat it as a *Node*, and execute the path of *Branches* following that *Node*. The simulation must be able to either produce a new state from the execution of the *Branch* path, or decide that the *Branch* is impossible to perform.

All but the simplest simulations used by the *APSS* should consider terrain effects. Terrain representation is necessary for event resolution, route and travel time determination, and fuel or other resource consumption determination. A minimal representation would include elevation and GO / SLOW-GO / NO-GO [4] depiction of the terrain. The terrain fidelity can be as high as permissible for efficiency and timeliness.

A more sophisticated and flexible simulation would be able to handle decomposable events. Multiple levels of resolution will allow the *APSS* to adapt to time and system resource constraints. For instance, the *Planner* might ask the simulation to resolve a company breach operation. If the *Planner* requires more detail, the system should be able to individually resolve the support force engagement, the breach force execution, and the assault force. Similarly, the system should be able to resolve a battalion versus company event as four companies versus one company, four companies versus three platoons, or twelve platoons versus three platoons.

2. Discrete Event Simulation

The *APSS* prototype developed for this research uses discrete event simulation (DES) mechanisms in many ways. A DES is used as the user constructs plans to determine the results of entity interaction and ensure the constructed plan is valid. Similarly, a DES is used when *Branches* are created by the *Branches Generator* to determine a new *Planned State* at the conclusion of the *Branch*. A DES is also used in a 'playback' mode to determine and display the actions taken within *Branches* by building an event list for the display and executing it in accordance with a user-selected time scale. An external DES is used to stimulate the *APSS* by providing an *Actual State* of the military operation. Finally, the *Execution Monitors* use a DES to produce *Anticipated States* for comparison with *Planned States*.

3. Synchronized Simulations

One of the key issues in testing the *APSS* prototype is to ensure that it is correctly stimulated. For testing purposes, the *Actual State* of the operation is produced by an

external discrete event simulation representing the *World View*. Regardless of the time scale used by the *APSS* prototype (1:1 for actual operations, much faster for playback or review) the external simulation must remain synchronized. For instance, if the *APSS* is operating more slowly than the external simulation, it could be generating new *Branches* for *Nodes* that have already been passed in "the real world." To keep the *APSS* and the *World View* synchronized, a *Test Executive* controls both. The *Test Executive* provides the interface for the human tester to load the appropriate *Plan Description* into the *APSS* and the modified *Plan Description* into the *World View*. Also, the *Test Executive* is where the tester establishes the time scale for operation of both systems, and where the tester can start and stop operation. Behind the scenes, the *Test Executive* sends control messages and receives notifications from the two systems, allowing it to keep them synchronized.

F. Monitoring the Situation and Re-planning with Agents

One of the primary purposes of the Anticipatory Planning process is to restrict the size of the planning space. Rather than consider every possibility in the plan, the process favors planning in front of more likely paths through the plan tree, and allows unlikely or impossible paths to be pruned away. The *Planning Executive* is responsible for restricting the consideration of alternatives and the creation of new *Branches*, and for identifying and pruning useless *Branches*. To accomplish this, the *Planning Executive* uses *Execution Monitors* to compare the *Actual State* to various *Planned States*, and *Planners* to produce new *Branches* as appropriate.

1. Planning Executive

The mission of the *Planning Executive* is to control the overall operation of the *APSS*. The *Planning Executive* creates and dispatches *Execution Monitors* and *Planners*. The *Planning Executive* controls how many *Execution Monitors* and how many *Planners* are operating at any time, sets the maximum branching factor at any *Node*, and tracks the state of the (computer) system on which the *APSS* is running.

When an *Execution Monitor* determines that re-planning should be conducted at a given *Node*, the *Execution Monitor* gives the *Planning Executive* a handle to the *Node* in question and a certainty associated with its recommendation. The list of *Nodes* for which re-planning is required as well as those *Nodes* at which re-planning is currently being conducted is called the Planning Frontier (see Figure 2). *Nodes* to the right of the frontier in the figure have been nominated for re-planning by an *Execution Monitor*, and *Nodes* to the left of the frontier have not been nominated.

The *Planning Executive* uses the confidence measures provided by *Execution Monitor*s to determine which *Nodes* along the frontier will get *Planners* allocated to them and in what order they will be allocated. If the *Planning Executive* decides that further planning is required for a *Node*, a *Planner* is launched and given the state (*Planned State* or *Anticipated State*) of the *Node*. The *Planner* examines the outcomes of different possible actions. If the system is very busy, the *Planning Executive* may determine that it can only afford a small number of running *Planners* and so *Planners* will have to be allocated to *Nodes* sequentially based on the criticality of creating new

Branches from the Node. If, however, the system is not busy, the *Planning Executive* may determine that it can afford to allocate a *Planner* to each *Node* along the frontier.

Similarly the *Planning Executive* determines how many *Execution Monitor*s are running at any given time. Again, if the system resources are not heavily used, the *Planning Executive* might put separate *Execution Monitor*s on many *Nodes*. On the other hand, in a resource-constrained situation, the *Planning Executive* might have only a few *Execution Monitor*s that hop from *Node* to *Node* under the control of the *Planning Executive*.

The *Planning Executive* also receives inputs from the interface with the user. Through the interface, the *Planning Executive* allows the user to manually insert *Branches* or to override work being done by *Execution Monitor*s or *Planners*. For instance, the commander might have some alternative action in mind and want to do a "what-if" analysis on it. Through the interface and *Planning Executive*, this new *Branch* could be added to a *Node* and a *Planner* launched. The *Planner* will complete the planning and determine that *Branch*'s viability. The commander might also want to manually delete a *Branch*, for whatever reason, and this is also done through the *Planning Executive*.

Finally, in a resource-constrained or very dynamic environment, it is possible that the creation of many *Branches* will exhaust available memory. In this case, the *Planning Executive* can set the maximum branching factor at *Nodes* to some small number (e.g., five). Thus, only the five most-viable *Branches* would be retained; other, less-viable *Branches* would be pruned.

The level of autonomy of the *Planning Executive* is a tunable parameter. It is likely that the intuition of some commanders might be a better predictor of a *Branch*'s viability than the decision of a *Branch Evaluator.* The user, therefore, might want to confirm the removal of all *Branches.*

By performing the actions described, the *Planning Executive* helps limit the scope of responsibility of the *Execution Monitor*s and *Planners.* The *Execution Monitor*s and *Planners* do not need visibility of the global state of the plan or the planning frontier. They merely need to know how to conduct their analysis or planning, respectively. This makes the job of designing and implementing *Execution Monitor*s and *Planners* much more tractable. When a *Planner* is dispatched, it must be provided a handle to the *Node* in question and the mission/objective of the operation. An *Execution Monitor* only needs to know the *Node* - and its associated state - that it is supposed to monitor.

2. Execution Monitors

The purpose of the *Execution Monitor* is to detect divergence of the operation from the *Planned States* that make up the *Plan Description. Execution Monitors* have access to the *Plan Description* as well as the *Actual State* of the operation. The *Planning Executive* can re-assign an *Execution Monitor* to monitor another *Node*, but the *Execution Monitor* is only concerned with one *Node* at any given time.

When the *Planner* builds the various *Branches* from a *Node*, it also creates an initial *Planned State* of the operation at that Node. The function of the *Execution Monitor* is to periodically produce an *Anticipated State* by forward simulation from the

Actual State of the operation to the *Planned State* held by a *Node*. An *Execution Monitor* must infer when the *Anticipated State* of the operation differs "significantly" from the *Planned State*. When significant differences occur the *Execution Monitor* performs several important tasks.

First, it conducts a breadth-first traversal of the *Plan Description*. At each *Node* in the *Plan Description*, the *Execution Monitor* determines whether the change in state invalidates any *Branches* leaving the Node. Recall that in the *Plan Description* preconditions are associated with each outgoing *Branch* from a *Node*. When the differences between the *Anticipated State* and the *Planned State* indicate that conditions associated with a *Node* cannot be met, that *Branch* (and all following *Nodes* and *Branches*) may be pruned.

Second, after the identification of prunable *Branches* has been completed, the *Execution Monitor* must determine whether there are "enough" viable *Branches* from the state. A *Planner* has previously determined the viability of the *Branches*. While the exact computation will be determined as part of this research, the *Execution Monitor* will use the number of *Branches* as well as each *Branch*'s viability to determine whether it thinks a *Planner* is needed to generate more options for the human user. If the *Execution Monitor* thinks that there are insufficient *Branches* from a *Node*, the *Execution Monitor* makes a recommendation to the *Planning Executive* with some measure of confidence. It is then up to the *Planning Executive* to allocate a *Planner* to the *Node* (as discussed previously).

In addition to comparing the *Anticipated State* to the *Planned State*, the *Execution Monitor* also looks at all conditions associated with the *Node*'s *Branches*. The *Execution Monitor* periodically checks each *Branch*'s conditions and looks at the *Actual State* of the operation. If something necessary to fulfill a condition is eliminated (e.g., a mine-clearing device has been destroyed or an infantry company has been wiped out) the *Execution Monitor* must notify the *Planning Executive* that the *Branch* should be considered for pruning.

Although it would be tempting for the *Planning Executive* to eliminate *Branches* that cannot be reached, this must be done with care. It may be possible that some event in a *Node* closer to the trunk of the tree will allow the condition to later be met. On the other hand, the *Planning Executive* should automatically prune *Branches* associated with conditions that can never be met, such as the destruction of a bridge or dam. *Branches* associated with conditions that might conceivably be met in the future should be retained. For instance, a battalion might receive another mine clearing device, replacement unit, sortie of close air support, or other assets from a higher headquarters. When a "recoverable" condition cannot be met, the *Execution Monitor* should notify the *Planning Executive*, so that the *Planning Executive* can notify the user. If the *Execution Monitor* is monitoring a *Node* sufficiently far into the future, it might be possible for the user to take an action that will allow the condition to be met.

Surdu, Haines, and Pooch [7, 14] developed a system called OpSim, designed to monitor the current operation. The result of that research verified the feasibility of *Execution Monitor*s as described here. OpSim uses a dynamic hierarchy of rational

agents, called Operations Monitors to compare the current situation with the plan. The top-level Operations Monitor informs the decision maker when the success of the plan is at risk. OpSim, or a system like it, could be adapted for use as an *Execution Monitor.* When OpSim was developed, the *Plan Description* described in this research did not exist. OpSim could be modified to access and understand the *Plan Description.* Then in addition to the inferences it makes based on state information, it could also look at whether conditions associated with *Nodes* can be fulfilled.

3. Planners

The *Planner* receives a state (*Planned State*, *Anticipated State*, or *Actual State*) and a mission/objective from the *Planning Executive.* The *Planner* invokes a *Branches Generator* and passes it the state and mission/objective. The *Branches Generator* returns some number of *Branches* to the plan, along with their associated preconditions and confidence measures. At the end of each *Branch* is a new *Node* and the *Planned State* that the *Planner* predicts will exist after that *Branch* is followed. In an unconstrained environment, the *Planner* continues to execute a *Branches Generator* at each newly created *Node* until either the desired end state is reached or the *Branches Generator* determines that the desired end state cannot be reached. The *Planning Executive* can place constrains on the *Planner* that limits the planning in terms of time, depth, system resources, etc.. A *Branch Evaluator* evaluates each *Branch* and returns a viability measure.

If the *Planner* is operating on a *Node* with existing *Branches* (i.e., the *Node* has already been run through a *Planner*, but has been identified by an *Execution Monitor* as

needing further planning), the *Planner* compares the newly generated *Branches* to the existing *Branches*. If a new *Branch* is the same as an old *Branch*, the old *Branch* can be considered revalidated. If an old *Branch* is not revalidated based on the Anticipated State, the *Planner* notifies the *Planning Executive* that the *Branch* may be considered for pruning.

After the *Planner* is finished, the new *Nodes* at the end of the *Branches* may or may not be explored further. It is up to the *Planning Executive* to decide whether to place *Execution Monitors* on those *Nodes* and whether to act on any recommendations from the *Execution Monitor*s for further planning.

4. Branches Generator

The *Branches Generator* receives and examines a state and a mission/objective, then uses inference systems to generate different options. Prototype systems such as Fox-GA [157], Tactical Event Resolution [168], and the modified version of ModSAF used by Porto, et al. [165] have demonstrated the feasibility of automatic generation of courses of action in the military domain. The output of the *Branches Generator* is some number of distinct *Branches*, the *Planned State* that will hold after the action, and the associated confidence measures. The new *Planned State* will contain differences in the conditions of the entities (battle damage, destruction) and in resource consumption (ammunition, fuel, time).

To create new *Branches* the *Branches Generator* uses a genetic algorithm that starts with a user-definable number of initial random *Branches*. The algorithm uses a niching strategy in which the *Branches* of the first generation are created by heuristics

that tend to lead friendly forces to the desired friendly end-state and enemy forces to the desired enemy end-state. The initial generation is then run through a cycle of fitness testing and production of the next generation with a higher probability of reproduction for the *Branches* that have a higher fitness. Crossover is achieved by replacing the task-list for a specific entity in one *Branch* with the task list from the other *Branch*. Mutation is accomplished by creating a new heuristically-guided random task list for a particular tactical entity in the *Branch*.

5. Branch Evaluator

The *Branch Evaluator* is given a *Branch* to evaluate and the desired friendly and enemy end-states. The *Branch Evaluator* compares the *Planned State* at the end of the *Branch* with the desired end states of the operation, then uses an inference mechanism to determine the feasibility, acceptability, and suitability of the that *Node* (i.e., its viability). If the plan is in danger of failure (from the friendly perspective) at the new state, the *Branch* is assigned a low viability measure. If there is little danger of failure, the *Branch* is assigned a high viability measure. These viability measures are first generated at the leaves and propagated back up the tree. *Execution Monitors* use this viability measure when they analyze *Nodes*.

CHAPTER IV

IMPLEMENTATION

A. Introduction

In order to confirm that the Anticipatory Planning concept can be successfully supported by automated systems, an *Anticipatory Planning Support System* (*APSS*) prototype has been implemented. A testing suite has also been implemented, and is described in Chapter V.

From the outset, the implementation has been designed to provide the maximum visual interaction between the user and the system. However, since the underlying data structures are intended for use in later systems it is important to separate the information contained in the system from the visual handling of that information. Consequently, the system is composed of data elements and visual components that incorporate those data elements.

Although the environment being modeled (battlefield operations) is continuous, the implementation relies on discrete changes in state. The use of discrete representations simplifies some of the more complex problems, particularly those that are not important in the evaluation of the methodology. For example, the discrete hexagonal representation of terrain simplifies the placement and movement of tactical entities, and the interactions between them.

The Java programming language was used to implement the *APSS* and the testing suite. The system has been purposefully implemented to isolate and encapsulate the data

and functionality that belong together into the smallest possible objects. This enhances reusability of the objects, which makes it possible to combine them in many different ways for different purposes with little additional coding. A particular advantage to using Java (and other object-oriented languages) is that objects can be extended to add new functionality without having to modify the utility of the underlying class.

This chapter discusses issues in verification and validation, then describes the entities involved in planning, how plans are represented, displayed and built, how agents are used to monitor the plan and control planning, how attrition is modeled, and how discrete event simulations are used to determine the results of entity interactions.

B. Verification and Validation

The literature review in Chapter II revealed three techniques for validating models and eight techniques for verifying systems, in particular those due to Law and Kelton [8]. Refer back to Table 1 for a list of the validation techniques, and to Table 2 for a list of the verification techniques. Where appropriate, specific details of verification and validation are mentioned in the sections below. In general terms, several of the techniques have been enforced by the nature of the prototype system.

Verification technique number one requires the use of modular programming to narrow the scope of responsibility and difficulty into small, easily verifiable pieces. Every set of data and actions that can be logically combined into a Java object has been. This approach greatly eased the process of verification (and by extension, debugging) since the responsibility of each object is so well and narrowly defined.

Verification technique number six encourages the use of animations to make it simpler to identify failures in a system. From the outset of development the prototype has been heavily oriented on providing a graphical user interface for every step of entity building, plan building and display, and test operations. This has the intended effect of making the inner workings of the prototype system completely visible to the user. Consequently, verification of the system operation was made much simpler.

Verification technique number eight suggests the use of pre-existing simulation systems or packages to reduce development time and prevent reinvention of common algorithms and approaches. While generally promising, this technique runs the risk of using systems with embedded errors or inefficiencies that cannot be accounted for in the prototype system. To ensure that all anomalies can be properly attributed and corrected, the prototype system and its discrete event simulation systems have been implemented completely in Java, rather than using pre-existing simulations.

Validation of the system relies on three major techniques. First, face validity is obtained by placing experts in the military planning domain in front of the system and gathering their assessments of how well the system models that domain. Second, empirical testing of the assumptions used by the system has been performed around each module that relies on those assumptions. The third validation technique involves determining how representative the output data of the system are. Since there is no existing anticipatory planning system in which branches in a plan are produced and pruned under the control of cooperating agents, it is impossible to validate the *APSS* prototype against other planning systems. Rather, the approach from the first technique

used to determine face validity is also used to determine whether the outcome of the system (focused planning effort) is valuable to military planners.

C. Entities

There are two fundamental types of entities used in the *APSS*. The first is tactical entities representing military units, obstacles, and the effects of indirect fire. The second fundamental type is terrain, which affects the movement, target acquisition, and other activities of the tactical entities.

1. Tactical Entities

Tactical entities are represented in the system as instances of the TacEntity class. Multiple TacEntities can be organized into a force, and are held in a TacEntityTreeModel that identifies the subordination relationships between the tactical entities. As the plan is built, specific values of TacEntity attributes at critical plan states, called Nodes, are stored in the TacEntityState class. Specific transitions or activities of the TacEntities between the Nodes are stored in TacEntityTasks. As TacEntityTasks are performed, changes in the status of the TacEntities are stored in TacEntityStatusChanges. Finally, there are a number of visual components that allow for interactive manipulation of the tactical entities.

a. TacEntity (Class)

Every TacEntity has a globally unique, persistent identifier. This is used rather than pointers to objects to allow for persistent plans. It also enables future

implementations of the system with a supporting database. The TacEntities also have a name for more intuitive identification by the user.

TacEntities fall into three basic categories: units, obstacles, or effects. They are affiliated with the enemy force, the friendly force. They may also be neutral, or have unknown affiliation.

Units have a level that identifies how big they are, ranging from individuals through division-size units. They also have a role that indicates their capabilities (armor, mechanized infantry, engineer, etc.). If the TacEntity is an obstacle, it can be one of two types: linear or area. Also, if it is linear, it has a direction associated with it.

TacEntities have several attributes used for initialization: beginning strength, maximum fuel, maximum ammo, etc. They also have variable attributes that make up their state at any given time, including location, current strength, current fuel, etc.

b. TacEntityTreeModel (Class)

The TacEntityTreeModel is used to manage the tactical entities that will be used by the *APSS* system. It serves as the central repository for holding the TacEntities. It also maintains the subordination relationship information. There is only one instance of the TacEntityTreeModel when the system is running. This means that there is only one copy of each TacEntity. This ensures consistency whenever an update is made to the TacEntity, and it enables future database implementation of the system. All of the TacEntities that may participate in a planning session must be in the TacEntityTreeModel at the beginning of plan development.

c. TacEntityState (Class)

The TacEntityState holds the relationship between a TacEntity and values assigned to its attributes, such as its location and its strength at a particular time in the plan. It is important to note that the TacEntityState does not itself know the plan time it is associated with. The TacEntityState for all of the TacEntities are held in a plan node (described later), and it is this node that keeps track of their temporal location.

d. TacEntityTask (Class)

Actions taken by the TacEntities are described in the system by instances of the TacEntityTask. There are several types of TacEntityTask, including idle tasks, movement tasks, activation tasks (for artillery bursts) and buildup tasks (for obstacles). The TacEntityTask defines the current and next location of its associated TacEntity to enable the system to ensure consistency from task to task. It also retains the duration of the task, allowing the system to place future tasks at the correct time.

e. TacEntityStatusChange (Class)

As the tactical entities perform tasks, the interaction between entities produces changes in their status. Note that the word status is deliberately used instead of state. In this system, important states are held by the plan nodes. The changes in attributes between the states are tracked separately as status changes. A TacEntityStatus change includes information as to the type of change (strength, fuel, ammo) and the change factor, a variable from 0.0 to 1.0 indicating the amount left out of what was available before the change.

f. TacEntityComponent (Visual Component) (Drag and Drop)

For visualization purposes and to ensure separation between the data and the interfaces of the system, TacEntities are wrapped in a TacEntityComponent that handles all of the drawing functions, handling of mouse click events, and drag-and-drop operations. The TacEntityComponent also handles the visual interactions with the visual terrain components. See Figure 3 for examples of a TacEntityComponent held by a TacEntityDragPanel, and the TacEntityCompnent class drawing icons for the labels of the TacEntityTree. Also, see Figure 7 for examples of TacEntityComponents displayed on a HexGridPlanPanel.

g. TacEntityDragPanel (Visual Component) (Drag and Drop)

The TacEntityDragPanel displays a TacEntityComponent. Any changes made to the underlying TacEntity are instantly visible in the TacEntityComponent. The purpose of the TacEntityDragPanel is to allow the user to click on the TacEntityComponent and drag it to any visual component that accepts TacEntity drops. See Figure 3, where a TacEntityDragPanel appears in the upper-left corner.

h. TacEntityConfigPanel (Visual Component)

The TacEntityConfigPanel holds a TacEntityDragPanel and several combo boxes enabling modification of the TacEntity's attributes, such as category, affiliation, role, level, etc. Changes in the selections in the combo boxes are instantly applied to the TacEntityDragPanel. See Figure 3, where a TacEntityConfigPanel occupies the left-center portion of the TacEntityBuilder.

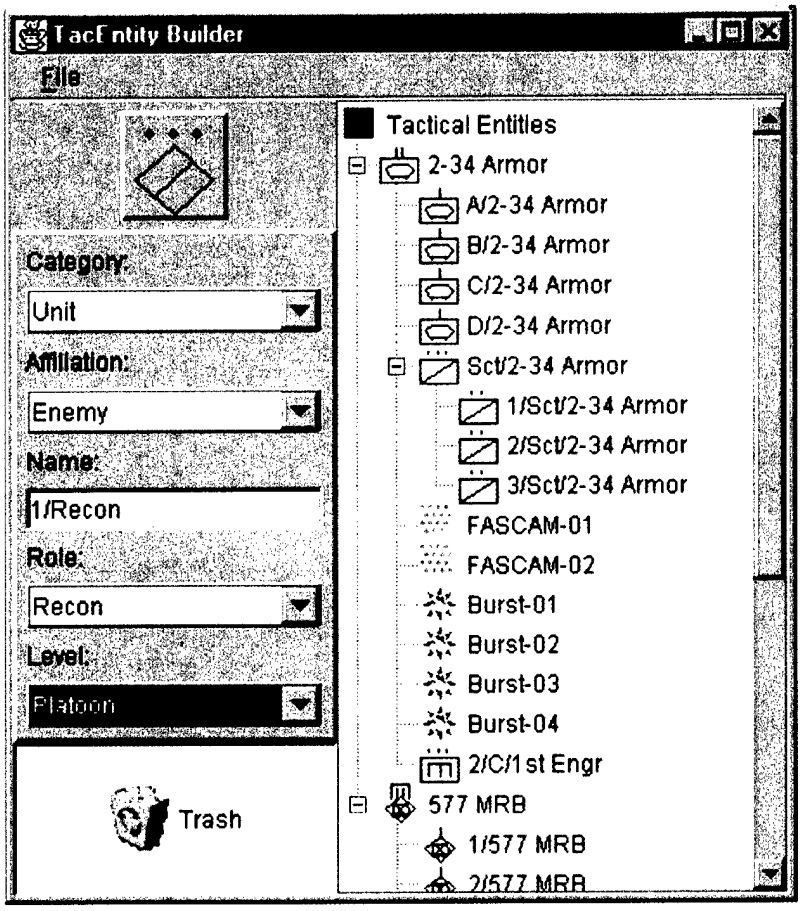

Figure 3: TacEntity Builder application showing several visual components

i. TacEntityTransferable (Drag and Drop)

In drag-and-drop operations the TacEntityTransferable represents the TacEntity while it is in the process of being dragged. When a drop is attempted, the TacEntityTransferable is passed to the visual component that is the target of the drop. The TacEntityTransferable contains information that allows the drop target to determine if it will accept the drop, and if so passes it the necessary information. Note that this is a very important reason to have TacEntities identified by a globally unique identifier. When the drop operation completes, a new copy (effectively a clone) of the dragged

object is instantiated. Any objects that point to the original TacEntity do not point to the new instance. However, the new instance does include the unique identifier. So long as all references to the specific TacEntity are made through this identifier, there is no confusion.

j. TacEntityTree (Visual Component) (Drag and Drop)

The TacEntityTree takes the information from the TacEntityTreeModel and visually displays the TacEntities and their subordination relationships. It also allows for visual drag-and-drop changes of those relationships. The TacEntityTree can accept drops of TacEntities, either from TacEntityDragPanels or from other TacEntityTrees. See Figure 3, where a TacEntityTree occupies the right half of the TacEntityBuilder. Although not important in this implementation, units of different roles (specifically, armor and infantry) that are combined under a headquarters cause that headquarters to display a "task-organized" indicator.

k. TacEntityBuilder (Application)

The TacEntityBuilder application allows for the interactive construction of individual TacEntities and for their hierarchical arrangement. See Figure 3 for a screen capture of the TacEntityBuilder in the process of building the TacEntityTreeModel to be used in a planning session. A TacEntity is built by selecting its attributes, causing the icon representation to change appropriately. Once the TacEntity is complete it can be dragged and dropped onto the hierarchical TacEntityTree. Note that when the TacEntities are dropped on the TacEntityTree is when the persistent, globally unique identifier is assigned. If that TacEntity is subsequently removed from the TacEntityTree

(really, from the underlying TacEntityTreeModel) its identifier is discarded and never used again. The TacEntities can be moved around within the TacEntityTree so long as the new ordering is consistent with certain rules. For example, friendly units must belong to a friendly headquarters and smaller units must be underneath larger headquarters.

2. Terrain

For this implementation of the *APSS* prototype, terrain is represented by hexagonal cells (HexCells) arranged in a rectangular grid (HexGrid). The use of cells allows for discrete changes of location, and localizes information about terrain into defined regions. HexCellComponents are used to display the data held by the HexCells, and serve as a base class for handling mouse events and drag-and-drop operations. There are several varieties of panels used for displaying and manipulating HexGrids. A HexGrid Builder has been implemented to simplify the construction and specification of HexGrids.

a. HexCell (Class)

Attributes of the terrain are held in various 'styles' within the HexCell. For instance, trafficability is represented by the goStyle and is implemented in integer steps. A HexCell that gives no hindrance to travel is described as "Fast-Go", while "No-Go" terrain seriously hinders travel. Similarly, the amount of vegetation in the cell is represented by a scale from "No-Veg" through "Heavy-Veg." The amount of buildings and other man-made structures is indicated by the builtupStyle, ranging from "No-Builtup" through "Heavy-Builtup." The effect of the terrain on the TacEntities is

calculated using the value of each of these styles. For instance, the goStyle is used in an exponential function, causing increasing movement delays for increasing difficulty in trafficability.

The edges of HexCells have attributes as well. Edges are used to represent streams or rivers ("Water-Edge"), or to show changes in elevation ("Contour-Edge"). When traversing the cell, either from the center to an edge, or vice-versa, the presence of a road is indicated by a roadStyle ranging from "No-Road" through "Heavy-Road."

Each HexCell holds pointers to its six neighboring cells. This allows for rapid selection of nearby cells without having to go through the HexGrid array. Also, HexCells are identified by an instance of the Location class. Locations have an X and Y coordinate, and a number of utility functions, such as finding the distance between two Locations.

b. HexGrid (Class)

The HexGrid class holds an arrary of HexCells, which allows for rapid retrieval of the HexCells by just knowing their Location. A movement task contains an identifier for the TacEntity making the move and the Locations of the current HexCell and the next HexCell. The HexGrid is the only object that can retrieve the actual HexCells referred to by the Locations. As such, the HexGrid is the object that calculates duration of movements.

c. HexCell Components (Visual Components)

The HexCellComponent base class provides the means of visualizing and interfacing with HexCells. See Figure 4 for an example of HexCellComponents on a

HexGrid Panel. The class has a number of constants representing the colors to be used for various attributes of the terrain, such as the shade of green to use for vegetation. The Hexagon class is used to define the outline of the HexCellComponent when it is visible. The HexCellComponent knows how to add itself to and remove itself from a HexGrid Panel (see below). It also knows how to hold and display a TacEntityComponent that has been placed on it. Finally, it knows when the user has clicked the mouse on it, and it can determine which edge the mouse-click is closest to. This is important in editing the HexCell.

The DropTacEntityHCC is a sub-class of HexCellComponent that knows how to accept TacEntityComponents that have been drag-and-dropped onto it. This class is only used in the HexGridPlanPanel to enable visual construction of plans.

d. HexGrid Panels (Visual Components)

The HexGridDisplayPanel is the super-class for all panels that display HexGrids. It holds the fundamental data, such as an array of HexCellComponents that is isomorphic to the HexCell array in the HexGrid. It also handles all the routine operations of opening, loading, and saving HexGrids.

The HexGridEditPanel extends the HexGridDisplay panel to allow visual interaction for modification of HexCells in the HexGrid. See Figure 4 for an example of a HexGridEditPanel and a HexGrid under construction. This class handles the monitoring of mouse clicks and the modification of HexCell attributes. .

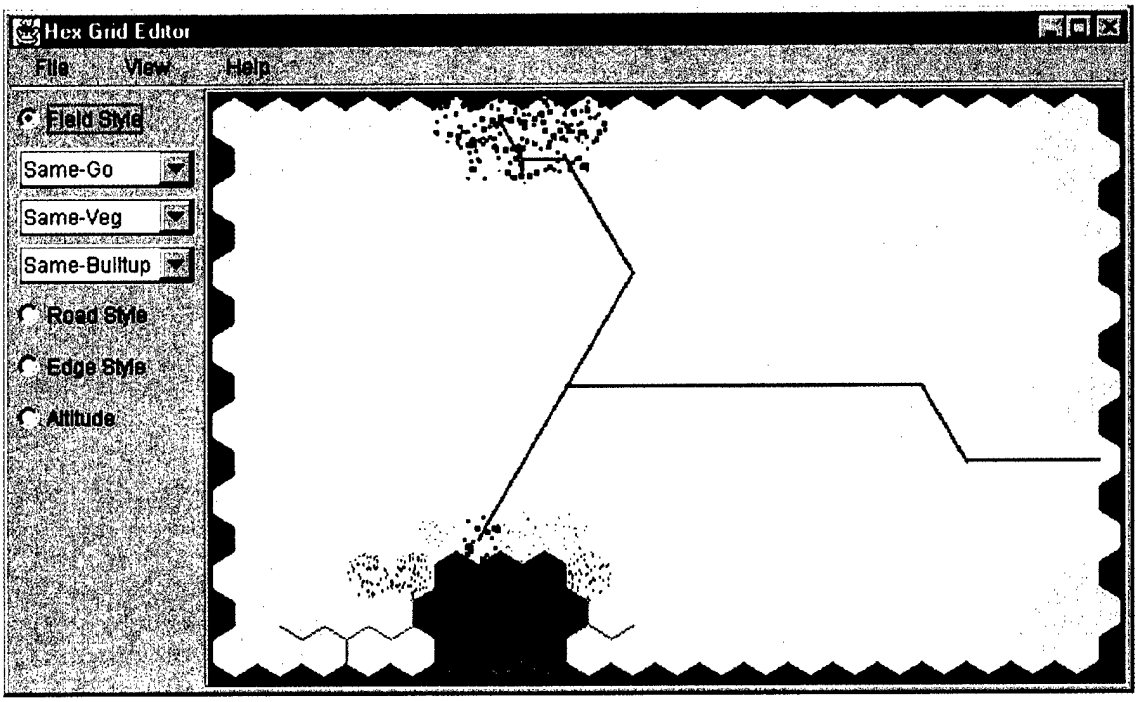

Figure 4: HexGrid Builder showing HexGridEditPanel and HexCellComponents

The SnapshotPanel, a separate sub-class of the HexGridDisplayPanel, is designed to display particular moments, or shapshots, in a plan. It keeps a HashMap of TaqEntityComponents that are currently displayed on the panel. If a TacEntityComponent is to be added, moved, or removed from the display, this makes it much easier and faster, since only the affected TacEntityComponents and HexCellComponents need to be redrawn. The SnapshotPanel can receive a Node, or a Branch with a specified current time, and display the state of the operation at that time.

The HexGridPlanPanel extends the SnapshotPanel for planning purposes. It allows the interactive placement and manipulation of TacEntities, and is discussed in section D.3 of this chapter.

e. HexGridBuilder (Application)

The HexGridBuilder was implemented to allow for rapid and interactive creation, editing, and storage of HexGrids. See Figure 4 for a screen capture. It combines a HexGridEditPanel and a HexGridConfigPanel (on the left side of Figure 4) and provides the necessary control functions.

D. Representing, Displaying, and Building the Plan

The basic idea for representing a plan is to think in terms of situations and transitions between those situations. Some of the situations are option points, where the human planner might think, "From here, we can do this, or we can do that." A PlanDescription composed of Nodes (option points) and Branches (transitions) is used to represent a plan.

All of the visual components in the system are designed for ease of reuse. Components are configured in different ways, depending on the desired process. An Executive handles the interactions between components. All of the components send messages to their Executive. Different Executives have been built to handle different configurations. Depending on their function, these Executives handle or ignore messages from the components.

To display a plan and allow for visual interaction, several visual components have been developed. Nodes and Branches have been wrapped in visual components. A PlanDescription Display Panel handles visualization of the plan and interaction with the

user. Internal plan information, such as the states within the Nodes and the transitions within the Branches, is displayed with different visualization mechanisms.

To allow the user to build a plan, several classes have been developed. A PlanBuilderPanel brings together panels for handling TacEntities, creating new Branches in the PlanDescription, creating states states, and assignment of tasks in Branches. A PlanBuilderExecutive controls the interface with the user and the plan building process.

1. Representing a Plan

The *Anticipatory Planning Support System* relies heavily on a common description of a plan. To represent a plan, a PlanDescription is dynamically built to manage the many tree-like branches that occur in planning and execution of an operation.

The word "common" is used to indicate that every major sub-system in the *APSS* makes use of the same PlanDescription. The human planner uses the GUI to modify a plan. Execution of the actual operation is itself represented by a PlanDescription, and is compared to the PlanDescription used within the *APSS*. The genetic algorithm operates on segments of the PlanDescription during re-planning. The simulations process segments of the PlanDescription to determine outcomes and provide evaluations.

The PlanDescription is composed of Nodes that hold information about the state for each situation, and Branches that hold the transition information between the states. Each Node may have zero or more Branches leaving from it. Thus, the PlanDescription is like a tree.

One advantage to the tree-like representation of a plan is that the transitions held by the Branches can be in relative terms. A change in the transitions of a Branch results in a new state in the Node at the end of the Branch. Following Nodes can be updated based on that change without having to modify the intervening Branches in any way. This makes for very rapid updates when a plan is modified.

a. PlanDescription (Class)

The PlanDescription class holds the HexGrid representing the terrain a plan is built on, and a TacEntityTreeModel that manages the TacEntities used in a plan. The PlanDescription only keeps track of three things: the root Node, the node that is currently in focus, and the currently selected Branch, if any. It's only other function is to open and save PlanDescription files.

b. Node (Class)

The Node class holds information about the state of a plan at a particular moment. It keeps a time stamp to identify that moment and a HashMap of TacEntityStates, that taken together make up the *Planned State*. If it is not the root Node, it keeps track of its previous branch. Since (so far) there has been no ordering or sorting requirement for the following Branches, they are held in a Vector.

The Node recurses through previous and following Nodes to retrieve timing information, such as the time stamp of the earliest or latest Node at a particular depth. The Node has the responsibility and capability for creation of new Branches. It also serves a very important function in updating the plan information. It can adjust its own time stamp, and cascade that change as an offset through all of its following Nodes. It

also serves as the focal point for propagating changes in the TacEntityStates held in the Nodes, also performed as updates or offsets.

c. Branch (Class)

Branches use HashMap of TacEntityTaskLists to manage the tasks assigned to the TacEntities that are present at that point in a plan. All of the TacEntityTaskLists taken together capture the transition from one option point in a plan to the next option point. They also contain a HashMap of TacEntityStatusLists developed by simulation of the interactions of the TacEntities. Branches keep track of their start Node and their end Node to serve as the connection between Nodes for information passing.

The Branch keeps track of the minimum duration of the combined TacEntityTaskLists. Whenever new tasks are added, or old tasks are modified or deleted, two things happen. First, all but the longest TacEntityTaskList is padded with a new IdleTask. This ensures that they are properly accounted for in TacEntity interactions. Second, a simulation is invoked to determine the interactions and build TacEntityStatusLists. Finally, the minimum duration is updated, if necessary, and the time change is propagated through all following Nodes.

d. TacEntityStatusList (Class)

The TacEntityStatusList class is a LinkedList of the TacEntityStatusChanges for a specific TacEntity. A LinkedList is used because the order of the status changes matters, rapid insertions and deletions are desired, and because rapid scanning of portions of the list is important. The list can be queried for a particular attribute of the

TacEntity at a given time since the previous Node. This information is used to update the situation on the SnapshotPanel and when simulations or playbacks are desired.

e. TacEntityTaskList (Class)

The TacEntityTaskList class is a LinkedList of TacEntityTasks for a specific TacEntity. A LinkedList is used because the order of tasks matters, rapid insertion and deletion are desired, and rapid scanning of portions of the list are important. This list can also be queried to gather specific information at a given time. The TacEntityTaskList remembers its own minimum duration, and transmits that information to its owning Branch. When changes occur to the list, it notifies its Branch so that appropriate action can be taken. The TacEntityTaskList performs all of the processing for insertion, modification, or deletion of a TacEntityTask.

2. Executives

The large variety of visual components developed for the *APSS* have been specifically designed to encapsulate their data and methods in a logical way. In this way, they can be combined to accomplish different purposes. When they are combined they must communicate with each other and their actions must be coordinated. To accomplish this coordination a number of Executives have been developed.

a. Executive (Class)

The Executive class serves as the super-class for handling all messages. It has empty methods that must be overridden by sub-classes when particular activities are desired. These activities include plan building, plan display, and simulation execution.

b. BranchScanExecutive (Class)

The BranchScanExecutive class is exclusively used to run rapid simulations of portions of a plan. Unlike the PlayerExecutive there is no requirement for time scaling. The BranchScanExecutive merely processes the simulation event queue as rapidly as possible to determine and handle all of the interactions between the entities.

The TacEntityTaskLists of the Branch are processed and events are loaded into the simulation. The events are processed without respect to a time scale, but in time stamp order. As each event is processed any existing interactions between TacEntities is resolved. The situation is then examined to either add new interactions, or to remove completed interactions.

For example, engagements between TacEntities are held in two HashMaps – one hashed by TacEntity ID for all of that TacEntity's targets, the other hashed by TacEntity ID for all of the other TacEntities that are shooting at that TacEntity. So long as neither target or shooter moves out of the shooter's range the engagement continues. Once they are out of the shooter's range the appropriate entries are deleted from the two HashMaps and the engagement is concluded.

Each time the engagement is resolved, a strength change is entered in the TacEntityStatusList for the appropriate TacEntity. Once the simulation has completed, the changes in end states for the TacEntities are propagated through the PlanDescription.

c. PlayerExecutive (Class)

The PlayerExecutive class is used when visualization of the flow of a plan is desired. Given a time scale, the PlayerExecutive sends control messages to a

SnapshotPanel telling it to display a succession of states in a plan. This has the effect of providing a 'playback' mechanism to the user. It is also used to provide an *Actual State* to stimulate the *APSS* during testing. The PlayerExecutive operates much the same way the BranchScanExecutive does. The distinction is that the PlayerExecutive uses the TacEntityStatusLists produced by the BranchScanExecutive and it operates in scaled 'real' time to provide realistic playback.

d. PlanBuilderExecutive (Class)

The PlanBuilderExecutive is a much more sophisticated sub-class of the Executive. It is discussed in more detail in the section on Building a Plan. It includes the capability of the PlayerExecutive and adds the ability to process user input for building a plan.

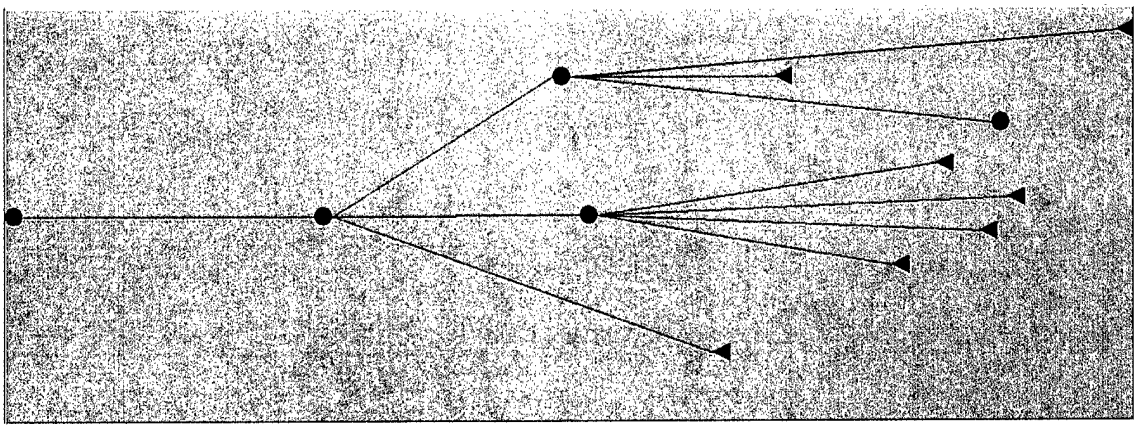

Figure 5: PDDisplayPanel showing BranchComponents and NodeComponents

3. Displaying the Plan

Nodes and Branches are not visible components; rather, they serve to hold the important data, make connections, and pass information. For visual display of a plan Nodes are represented by NodeComponents and Branches are represented by BranchComponents. These two components handle all of the interaction with the user (primarily mouse-driven). They send messages to the PDDisplayPanel that arranges the NodeComponenents and BranchComponents with respect to time. See Figure 5 for a screenshot of a PDDisplayPanel displaying NodeComponents and BranchComponents.

a. NodeComponent (Visual Component)

In the PDDisplayPanel, NodeComponents are shown as circles to indicate that they are expanded, showing their succeeding *Branches*. Alternatively, they are shown as triangles to represent the entire sub-tree from that Node and beyond. The NodeComponents can be selected with a left mouse-click, and a popup-menu can be invoked by a right mouse-click. The popup menu allows the user to modify a plan, such as creating a new Branch.

The NodeComponent knows how to display itself in the PDDisplayPanel. The NodeComponent is assigned a share of the y-dimension based on the number of children its parent has and its place out of the total number of its parent's children. It determines its X-position based on the proportion of its time stamp to the entire displayed time. Once it has displayed itself, it divides up its y-dimension display space among its children, then forwards the display information to all of its following NodeComponents so that they can display themselves.

b. BranchComponent (Visual Component)

The BranchComponent represents a Branch by drawing a line on the PDDisplayPanel between its start NodeComponent and its end NodeComponent. It listens for left mouse-click events indicating that the user has selected that Branch. It is colored red when it is the currently selected Branch.

c. PDDisplayPanel (Visual Component)

The PDDisplayPanel process a PlanDescription, produces NodeComponents and BranchComponents, and initiates the process of making the components draw themselves. If the PDDisplayPanel is resized, causing the drawing space to change, the image is refreshed so that it scales to the new drawing space. The PDDisplayPanel receives messages from the components indicating the user has taken some action, such as selecting a Node or Branch. It performs some internal processing, and then forwards the message to whichever Executive it is assigned to. To help the user focus on sub-trees in a plan, the depth (of the plan tree) can be controlled.

4. Displaying Branch and Node Information

In order to construct a plan and to visualize what is happening, the user needs a means of seeing the plan information; that is, the task lists held by the Branches and the states held by the Nodes. TaskDisplayLabels provide a visualization of individual tasks, and the TaskListDisplayPanel displays all of the task lists in a Branch. To display the state in a Node, the HexGridPlanPanel (descended from the SnapshotPanel) is used.

Table 3: TaskDisplayLabel Color Codes

Color Text	Color	Meaning
Yellow		Idle Task
Green		Move Task
Red		Active Task
Orange		Buildup Task
Blue		Filler Task

a. TaskDisplayLabel (Visual Component)

The TaskDisplayLabel class extends normal labels to provide visual indications of the attributes of its associated TacEntityTask. The label is color coded (see Table 3 for a description of the meaning of each color). The width of the label indicates the duration of the task, and is scaled to the minimum duration of the Branch (that is, the total displayed time). See Figure 6 for an example of the TaskDisplayLabels organized into a TaskListDisplayPanel.

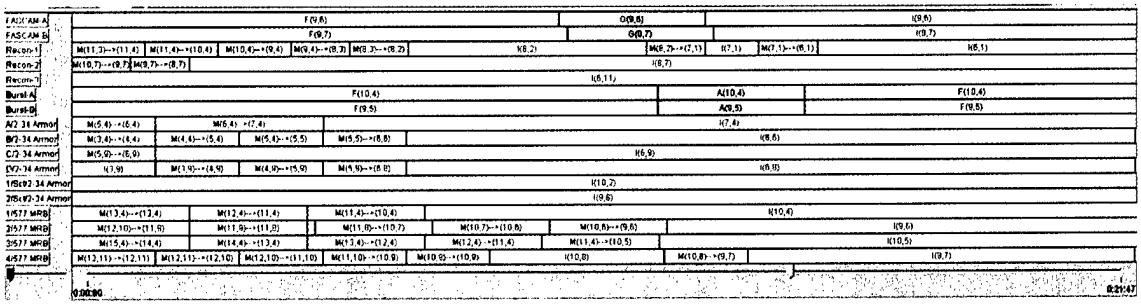

Figure 6: TaskListDisplayPanel showing TaskDisplayLabels

The TaskDisplayLabel listens for left mouse-clicks and left mouse-drags. This allows the user to reposition the task in time. So long as certain constraints are met, the user is allowed to move the TaskDisplayLabel earlier or later. It first checks its own

task, which can only be moved if it is not an idle or filler task. It then checks its neighbor tasks. If they are not idle tasks or filler tasks, then the TaskDisplayLabel cannot move in that direction. If it can move in one direction, it checks the task on the other side. If it is an idle or filler task, its start time and duration are adjusted as the TaskDisplayLabel moves. Otherwise, a new idle or filler task is created and inserted. It is then adjusted as previously described. Each TaskDisplayLabel listens for right mouse-clicks so that it can provide a popup menu. From this menu the user can remove the task or change its duration.

b. TaskListDisplayPanel (Visual Component)

The TaskListDisplayPanel organizes and displays the TaskDisplayLabels so that the user can view the internal workings of a Branch. See Figure 6 for a screenshot. On the left side is a panel displaying the names of the TacEntities involved in the Branch. At the bottom is a slider that allows the user to pick a particular time in the Branch and displays the minimum duration of the Branch. The main panel displays the internal information. If a Node has been selected, the main panel is blank. If a Branch has been selected, the task list for each TacEntity is presented as a sequence of TaskDisplayLabels. If the user attempts to make any modifications to the Branch, the TaskListDisplayPanel sends the appropriate notification or request message to the PlanBuilderExecutive. If the user resizes the panel, the entire TaskListDisplayPanel is adjusted to the new dimensions.

c. TacEntityDropHCC (Drag and Drop)

The TacEntityDropHCC is an extension of a normal HexCellComponent that handles attempts to drop TacEntityComponents. This extension is necessary for building plans with the HexGridPlanPanel. The TacEntityDropHCC doesn't actually accept the drop. Rather, it sends a message notifying the HexGridPlanPanel that the user has attempted a drop. If the PlanBuilderExecutive ultimately accepts the drop, the displays (including the HexGridPlanPanel) will be updated, otherwise no update occurs.

d. HexGridPlanPanel (Visual Component)

While building a plan it is important for the user to be able to see the state held within a Node or a particular time within a Branch. The user must also be able to enter changes to the plan. The HexGridPlanPanel is an extension of the SnapshotPanel with two modifications that are necessary for building plans. See Figure 7 for a screen capture of a HexGridPlanPanel displaying TacEntities.

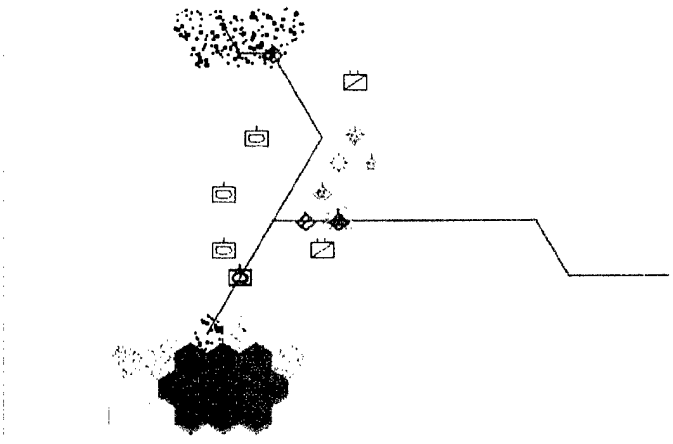

Figure 7: HexGridPlanPanel displaying TacEntityComponents

The first modification is that DropTacEntityHCCs rather than normal HexCellComponents represent the HexCells. The DropTacEntityHCCs intercepts attempts to drop TacEntityComponents and sends a request for permission from the HexGridPlanPanel to accept the drop.

The second modification is the addition of a method to receive the request message from the DropTacEntityHCC. The panel checks to see if that TacEntityComponent is already displayed, meaning that this drop is really an attempt to move that TacEntityComponent. If so, the panel sends a request to the PlanBuilderExecutive for permission to make the move. If not, it sends a request to add the TacEntityComponent. The PlanBuilderExecutive examines the current situation, attempts to create and insert the appropriate task(s), and replies with a Boolean granting or denying permission. If permission is denied, there is no change in the situation. If permission is granted, the situation displayed on the HexGridPlanPanel (and all other displays) is updated.

5. Building a Plan

The system allows the user to build a plan by providing all of the necessary interfaces, then managing their interactions with the user and with each other. The PlanBuilderPanel contains the user interfaces, and a PlanBuilderExecutive monitors the interactions, controls the displays, and decides which actions are allowed.

a. PlanBuilderPanel (Visual Component)

The PlanBuilderPanel provides all of the interfaces required by the user for building a plan. A TacEntityTree (see Figure 3) allows the user to drag-and-drop

TacEntities onto a HexGridPlanPanel. The HexGridPlanPanel (see Figure 7) displays the current state of a plan. A PDDisplayPanel (see Figure 5) lets the user navigate through a plan and modify it. A TaskListDisplayPanel (see Figure 6) shows the internal workings of the selected Branch. Finally, a DESimControlPanel provides an interface to a simulation for 'playback' purposes. All of the components send their request and notification messages to the PlanBuilderExecutive.

b. PlanBuilderExecutive (Class)

The PlanBuilderExecutive controls the execution of the plan building process. It starts by establishing contact with all of the subordinate display components of the PlanBuilderPanel and telling them where to send their messages. It then waits for the messages to arrive. If a 'node selected' or 'branch selected' message arrives, it sends control messages to all of the displays requiring them to update. If requests to add, modify, or delete tasks are received, it instructs the PlanDescription to make the attempt. Depending on the outcome, different control messages are sent out. If the user wants to run a 'playback' it loads the simulation and hands control to the DESimControlPanel, which merely forwards any instructions it receives from the user to the PlanBuilderExecutive.

E. Agents used in Monitoring a Plan and Controlling Planning

There are three types of agents used to monitor the execution of a plan, conduct re-planning, and control the process. The execution monitoring agents have the responsibility of monitoring Nodes, analyzing an ActualState, and assessing the

likelihood of their Node's occurrence. The re-planning agents examine the *Planned State* held within their Node and use the desired friendly and enemy end-states to produce new Branches. The PlanningExecutive agent receives timing and ActualState updates, assigns ExecutionMonitors to Nodes, and assigns Planners to future Nodes that are most likely to occur. The monitoring and re-planning process, and its associated time concerns, is represented in Figure 8.

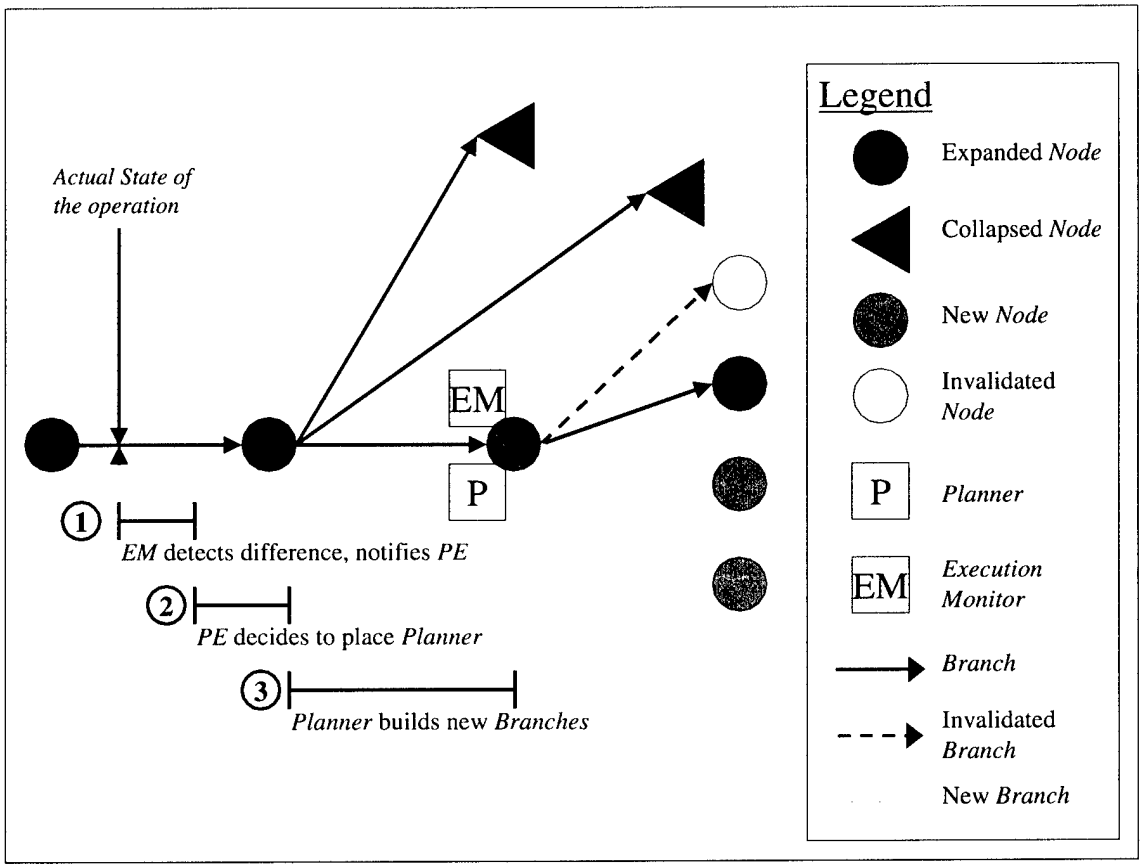

Figure 8: *APSS* Monitoring and Re-planning Process

1. The Execution Monitoring Agents

To detect deviations of the actual operation from a plan, instances of the ExecutionMonitor class are assigned to Nodes. These ExecutionMonitors periodically compare the ActualState to the state held by the Node and assign a likelihood measure to that Node. The likelihood measure is a function of the distances of all the TacEntities from their goal Location and the differences in their strengths, and is determined by a StateDifferenceAnalyzer. The ExecutionMonitor reports this difference to the PlanningExecutive. In Figure 8 an ExecutionMonitor has been attached to a Node, and the time window labeled ① represents the time used by the ExecutionMonitor to make its recommendation. Ideally, the ExecutionMonitors will complete their analyses and decide what recommendation to make before the next ActualState is available. Depending on the time scale in use, this is not always achievable. Currently, the ExecutionMonitors ignore any new ActualStates that arrive while they are processing. This information is not lost, however, since the WorldView keeps a copy of every published ActualState. If there is sufficient time, the ExecutionMonitors can try to catch up. Future implementations may reduce the fidelity and resolution of the underlying simulation in an attempt to complete the processing in time.

a. ExecutionMonitor (Class)

The ExecutionMonitor is a software agent that makes recommendations to the PlanningExecutive agent as to the likelihood of occurrence of a particular Node in the PlanDescription. It persists from the time of its creation by the PlanningExecutive until the Node it is observing has been bypassed in time. It can be reassigned to a new Node

if system constraints are such that object creation time is an issue. Typically, though, a new ExecutionMonitor is created and placed on newly monitored Nodes

The ExecutionMonitor really only has one important method, that of receiving a new ActualState and processing it along with its monitored Node through a StateDifferenceAnalyzer. Once the analyzer has been launched, the ExecutionMonitor stands by for the next ActualState.

b. StateDifferenceAnalyzer (Class)

The StateDifferenceAnalyzer runs as a thread so that all of the ExecutionMonitors can simultaneously run their own instance of it. Each StateDifferenceAnalyzer is given two states to consider. The first comes from the ActualState, the second from the Node being observed by the ExecutionMonitor. It is also given a RecommendationList into which it will post its recommendation. For every TacEntity, the differences in the two states of their Locations and their strengths are recorded. These are used in a weighted function to assess a likelihood measure (ranging from 0.0 to 1.0). This measure indicates how likely the *Planned State* held by the Node is to occur, given the current ActualState. Once the measure has been determined, it is sent to the RecommendationList.

c. RecommendationList (Class)

The RecommendationList is an extension of the LinkedList class that holds Node/likelihood pairs. The PlanningExecutive creates an instance of the RecommendationList whenever a new ActualState is received. It is given the number of ExecutionMonitors it should expect to get recommendations from, and then waits for

those recommendations. It has synchronized access methods so that the StateDifferenceAnalyzers can post their recommendations without causing consistency problems. When the last recommendation is received, it notifies the PlanningExecutive.

2. The Planning/Replanning Agent

To conduct re-planning, the re-planning agents are implemented as instances of the Planner class to generate and evaluate new Branches. A BranchesGenerator uses a genetic algorithm guided by the desired friendly and enemy end-states and inference mechanisms. The Planner uses a BranchesGenerator to consider possible friendly or enemy actions and produces significant, representative, Branches. The Planner invokes a BranchEvaluator to examine a Branch using simulation and inference mechanisms, and then determines a viability measure for the end Node of the new Branch. The viability measure indicates how well the state held in the end Node accomplishes the desired friendly end state.

In terms of timing, the Planner has at most until the ActualState reaches the Node to produce the new Branches. In Figure 8 time window ③ represents this maximum re-planning time. More often, however, a new ActualState arrives between the time of the ActualState and the time of the node being re-planned. The PlanningExecutive currently stops any Planners that are still working, on the assumption that the new ActualState will change the priority of re-planning.

a. Planner (Class)

The Planner class is responsible for the creation of new, representative, and significant Branches at a Node that has been designated for re-planning. It accomplishes

this by invoking a BranchesGenerator. When the BranchesGenerator has completed its execution of the Genetic Algorithm, the number of Branches specified by the user are pulled from the final generation and become the new Branches for the Node being re-planned. The end state of each new Branch is compared to the desired friendly end state with a StateDifferenceAnalyzer and given a viability measure. This viability measure is an estimate of how well the state at that Branch agrees with the desired end-state. The lower each of the three metrics (Location, strength, timestamp) is in relation to its equivalent in the desired end-state, the lower the viability measure.

b. BranchesGenerator (Class)

The BranchesGenerator class is a thread that handles the instantiation and operation of a Genetic Algorithm (GA) that does the work of producing new Branches. The GA has several parameters that govern its operation. The number of generations can be controlled, as well as the number of genomes in a generation. In this implementation, a genome is a Branch. The probability of crossover between mating genomes can be set, as can the probability of a mutation. Crossover is accomplished by swapping the paths taken by the same TacEntity in the two parent genomes. Mutation is accomplished by creating a new random path for the TacEntity.

The Genetic Algorithm uses a niching strategy based on the desired friendly and enemy end-states to create the first generation. The importance of the objective Location, desired end-strength, and desired end-time can be modified by adjustable weights. Also, the maximum time duration of the Branch can be set. Once each new

genome (Branch) has been created a BranchEvaluator is invoked to determine the state at the end Node.

c. BranchEvaluator (Class)

The BranchEvaluator is a simple class that is given a Branch and asked to determine the outcome. It uses a BranchScanExecutive to evaluate the interaction of the entities and the resulting state changes. This yields the new end state.

3. The Planning Executive Agent

The mechanism for controlling the planning and monitoring processes is embodied in the PlanningExecutive class. There is only one instance of the PlanningExecutive. It receives all information from outside the *APSS*, controls the assignment of ExecutionMonitors to Nodes, and controls the re-planning performed by the Planners. As each new ActualState is received, the PlanningExecutive adjusts the location of the ExeuctionMonitors, examines the recommendations from all of the ExecutionMonitors, determines the re-planning priority of the Nodes, and allocates Planners to the Nodes with the highest priorities. In Figure 8 the time window labeled ② represents the processing time between the PlanningExecutive receiving the recommendation from the ExecutionMonitors and allocating the Planners.

a. PlanningExecutive (Class)

The PlanningExecutive is initialized with a handle to the WorldView, from which it can draw information about the TacEntities and the HexGrid, and the PlanDescription. It immediately places ExecutionMonitors on all the child Nodes of the root Node in the PlanDescription, and stores the root Node as the lastActualStateNode.

It then waits for ActualState updates from outside the *APSS* (for testing, the ActualStates come from the TestExecutive).

When a new ActualState is received a new Node is created in the PlanDescription for that state, and the lastActualStateNode is updated to point to this Node. Also, an ExecutionMonitor is assigned to the new node. If any Planners are still running they are stopped, on the assumption that the new ActualState will cause different re-planning priorities. The time stamp of the ActualState is used to determine which Nodes in the PlanDescription have been bypassed. ExecutionMonitors are removed from bypassed Nodes and new ExecutionMonitors are placed on the nearest descendants of that Node that have not been bypassed. Finally, the new ActualState is sent to all the ExecutionMonitors for processing, and the PlanningExecutive stands by for the results.

When the RecommendationsList notifies the PlanningExecutive that all of the recommendations have been received, the PlanningExecutive examines the list and posts the new likelihood measures to the Nodes. It then requires all of the Nodes beyond the monitored Nodes to determine their re-planning priority. Each Node uses a weighted function of its likelihood, its time difference from the actual state, and the number of child Branches it has to determine its re-planning priority. It then adds itself to the planningPriorityList, which keeps the Nodes sorted from highest priority to lowest priority. Once all of the priority updates are posted the PlanningExecutive starts placing Planners on the Nodes in priority order. The Planners are allowed to run until

completion of BranchesGeneration, unless a new ActualState arrives, which cause the PlanningExecutive to terminate all of the still-running Planners.

F. Discrete Event Simulation

Discrete event simulation is used in several ways in the *APSS*. First, as the plan is being built the insertion, modification, or deletion of tasks can change the interactions within a Branch. Therefore, with every task change a simulation is run to determine the effects of the change. If the user desires a playback of a plan, a simulation is loaded with the task and state changes, then run at a user-selected time scale. When the ExecutionMonitors need to produce an *Anticipated State* they use a simulation. Finally, the Planners use simulations after a Branch is generated to determine the interactions and provide a *Planned State* for the end Node.

1. Simulation Mechanism

All of the simulations rely on a common discrete event simulation mechanism. This mechanism is composed of two major components. A DESimExec handles the execution of the simulation. A DESimEventQueue maintains the simulation events in time stamp order.

a. DESimExec (Class)

The DESimExec manages the various types of clocks involved in the simulation and processes the DESimEventQueue. There are two ways for this processing to occur: processing based on display time, and processing unrestricted by time concerns.

In the display time based processing, a tick clock is based on the system timer. The tick clock can be set to "go off" every 10, 100, or 1000 milliseconds. As each tick occurs, the DESimExec determines its relation to 'display time' (remember that the display may be playing back faster than real time). It then removes, in order, the events from the DESimEventQueue that have occurred since the last display time. Each of the events is executed, meaning that the appropriate Executive is notified that the event has occurred and what kind of event it is. The DESimExec then waits for the next tick to begin processing again.

The alternative means of processing is to step through the DESimEventQueue without respect to display time. This is appropriate when no playback is being provided to the user and only the fact that the events occurred at stated times is important. It is this processing means that makes it possible to detect and record engagements and other interactions in the BranchScanExecutive.

b. DESimEventQueue (Class)

The DESimEventQueue is precisely what it says it is: a queue of the simulation events. The time stamp of the events establishes their order in the queue. The DESimEventQueue provides the methods for inserting an event into the queue and popping the top event (lowest time stamp) off the queue. A LinkedList is used to implement the queue for several reasons. First, random insertions are not computationally expensive (linear, worst case is all the events must be considered). Second, it is possible for more than one event to have the same time stamp, which eliminates Sets or TreeSets as possibilities.

2. Simulation Events

There are several different types of simulation events. The base class for all events is the DESimEvent, which holds the time stamp of the event and an abstract Execute() method. All of the sub-class events must implement an Execute() method specific to themselves. For instance, the MoveEvent class sends a message to the appropriate Executive requesting that the TacEntity in question be moved to a new Location, while a StatusChangeEvent sends a request to modify one of the TacEntities attributes (strength, etc.) at the given time stamp. A DESimStoppedEvent is always the last event on the queue, and it notifies the executive that no more messages will be coming from the simulation, and it provides the Executive with the time stamp at which the simulation ended.

3. Controlling the Simulation

Control of the simulation is accomplished by using a common paradigm, that of starting, pausing, stopping, and rewinding a video stream. A TimeDisplayLabel shows the current display time of the simulation and the DESimControlPanel provides the buttons to accomplish each of the control steps.

a. TimeDisplayLabel (Visual Component)

The TimeDisplayLabel appears underneath the DESimControlPanel and shows the user the current display time within the simulation run. Before the simulation is started, it is passed a handle to this label. As the simulation processes its internal clock it updates the TimeDisplayLabel with the current display time.

b. DESimControlPanel (Visual Component)

The DESimControlPanel presents the user with several timing options for running the simulation. See Table 4 for a list of those options. This allows the user to run the simulation at anywhere from real time (1 sec = 1 sec) to a very fast time scale of one second equals one hour.

Table 4: Simulation Timing Options

Real Time = Plan Time
"1 sec = 1 sec",
"1 sec = 30 sec",
"1 sec = 1 min",
"1 sec = 2 min",
"1 sec = 3 min",
"1 sec = 5 min",
"1 sec = 10 min",
"1 sec = 1 hour"

There are four states for the DESimControlPanel, which correspond to the states of the simulation. See Figure 9 for the appearance of the control panel in each state. The simulation is in the 'ready' state when it is not currently processing a Branch. Once the user clicks on the 'Run' button, the simulation and the control panel transition into the 'running' mode, during which the only available option is to pause. The user may want to temporarily halt the simulation, so a click on the 'Pause' button puts the simulation into 'paused' mode, halting the processing of the event queue and displaying the state at that time. From the 'paused' state, the user can click the 'Run' button to resume processing, or the 'Rewind' button to return to the 'ready' state. If processing is resumed,

the system returns to the 'running' state. Once the event queue has been completely processed, the simulation enters the 'Stopped' state. This allows the user to see the situation at the end of the simulation run. From here, the user's only option is to press the 'Rewind' button to return to the 'Ready' state.

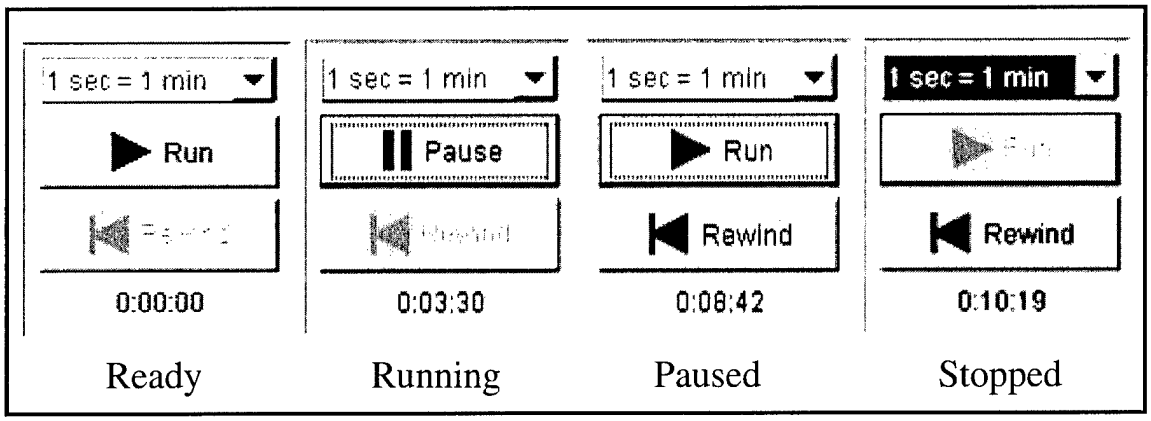

Figure 9: Simulation Control States

G. Attrition Modeling

The *APSS* uses a very simple attrition model based on relative strengths and time of engagement. See Figure 10 for a graph of the lossFactor applied after an engagement duration for odds ranging from 1:6 through 6:1. Clearly, future implementations of *APSS* will require more sophisticated attrition models.

As a Branch is scanned engagements are begun and terminated. Upon termination, the target is assessed a lossFactor representing the amount of their start strength that still remains. For instance, if the target starts at a strength of 1.2 and is being shot at 1:1 odds for three minutes, the lossFactor would be approximately 0.8.

This would reduce the target's strength to 1.2*0.8 = 0.96. Note that the target may also be simultaneously engaging the shooter; that engagement is resolved separately.

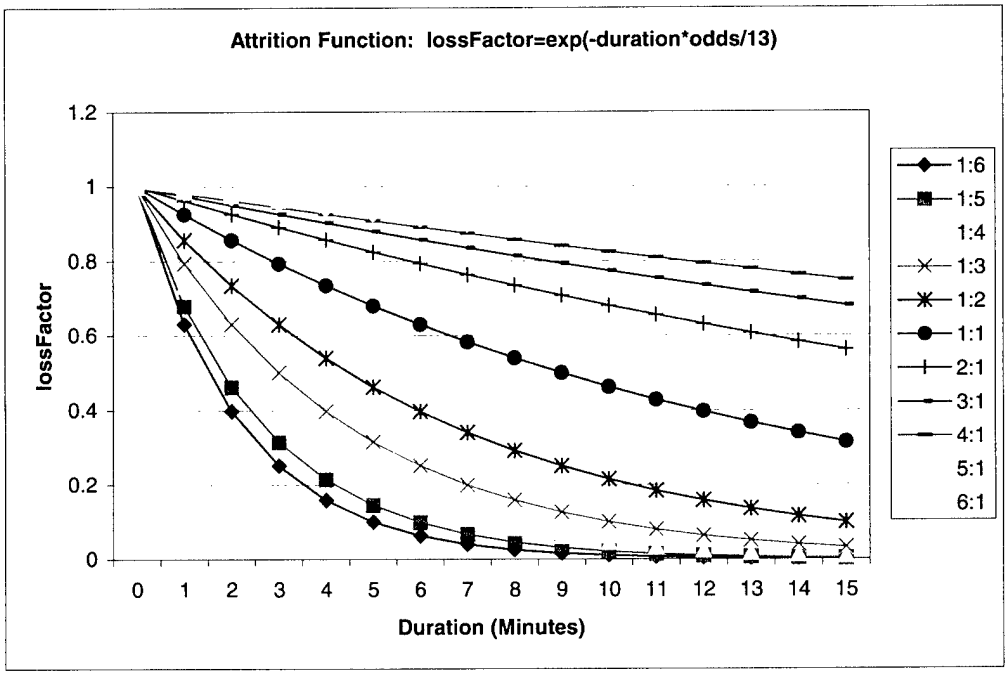

Figure 10: Attrition lossFactor as a function of duration and odds

The advantage to using lossFactors in this manner is that they are cumulative and translate well into status changes. In this way, sequence of status changes can be sequentially processed and the sequence of strengths easily determined and displayed. This is quite useful for interaction determination by a BranchScanExecutive and playback by a PlayerExecutive.

CHAPTER V

RESULTS AND ANALYSIS

A. Overview

This chapter contains the results and analysis of several tests that were conducted to verify the operation of the *Anticipatory Planning Support System* and to validate the underlying methodology. A comparison of the *APSS* to existing systems and research provides some context. The design of the system used to conduct experiments is described, and the parameters that can be varied for testing are identified. Three major tests are described, and the results analyzed.

B. Experimental Design

In a real military operation, automated command and control assets would provide the *Actual State* of the operation. For the purposes of testing, an external *Stimulator* that represents the activities of the *World Integrator* and *World View* components produces the *Actual State*. See Figure 11 for a depiction of the test setup. The *Test Executive* provides the interface for the human tester to load the appropriate *Plan Description* into the *APSS* and the modified *Plan Description* into the *Stimulator*. Also, the *Test Executive* allows the tester to establish the time scale for the operation, to start, pause, and stop the operation, and to post *Actual State* updates to the *APSS*.

The *Stimulator* operates on a test *Plan Description* that contains controlled differences from the *Plan Description* in the *APSS*. The *Stimulator* processes the test

Plan Description through a simulation to periodically produce *Actual States*. Meanwhile, the *APSS* is processing the *Actual States* on its own schedule. This can cause some timing concerns. For example, if the *APSS* is operating more slowly than the *Stimulator*, it could be generating new *Branches* for *Nodes* that have already been passed in "the real world."

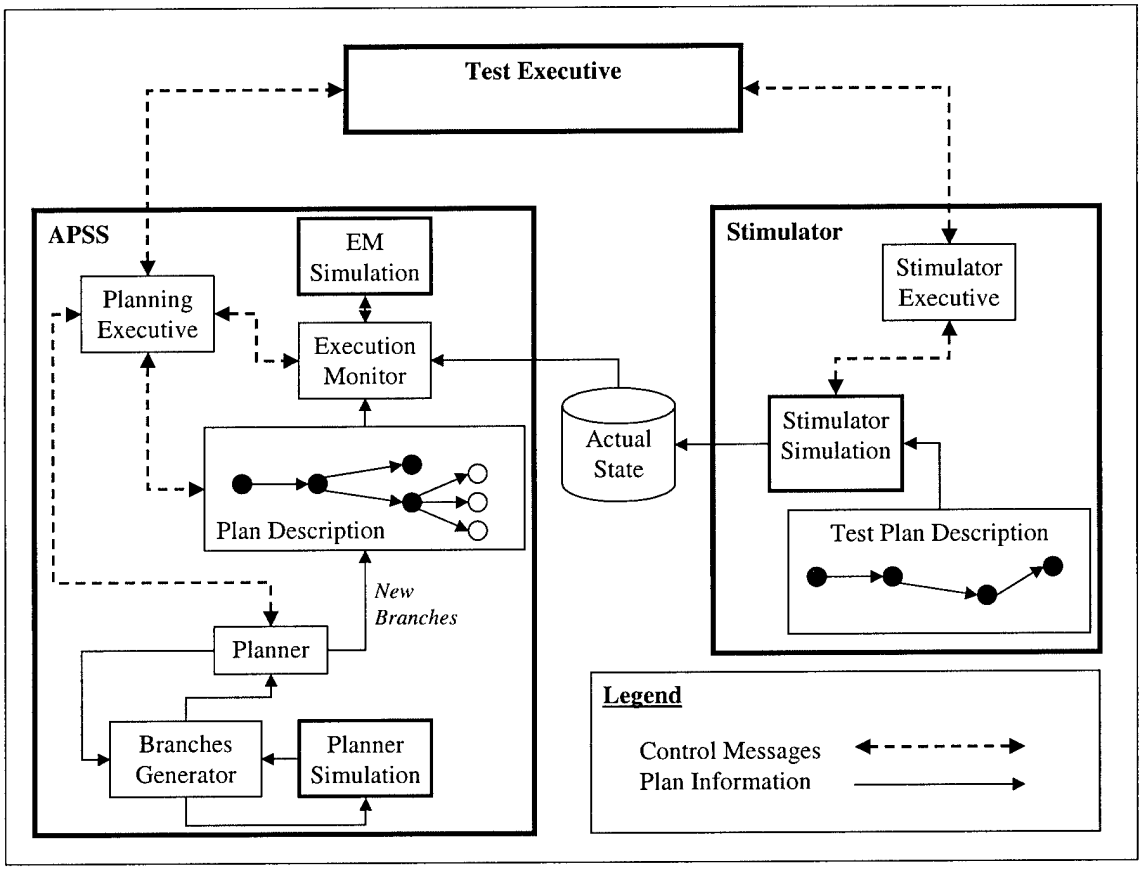

Figure 11: Testing System

The Test Executive keeps the *APSS* and the *Stimulator* synchronized by sending control messages and receiving notifications from the two systems. The *Test Executive*

maintains the master clock and the time scale. The *Stimulator* notifies the *Test Executive* when each new *Actual State*, containing a time stamp, is ready. At the appropriate time, the *Test Executive* sends a message to the *Stimulator* allowing it to post the new *Actual State*. The *Stimulator* replies with a confirmation of the posting, and then the *Test Executive* notifies the *Planning Executive* that a new *Actual State* is available.

C. Comparison to Existing Systems

There are no existing military planning systems that are structured or operate like the *Anticipatory Planning Support System*. See Chapter II for discussions of planning and COA production systems. There has been some work done on using a genetic algorithm to create courses of action (COAs). For example, the FOX-GA system [157] and the MEWS system [165] produce reasonable COA possibilities. The genetic algorithm used in *APSS* compares favorably to those systems since it produces reasonable COAs based on friendly and enemy strategies. None of the COA generation systems rely on monitoring of the situation to plan ahead of the flow of battle. Another fundamental difference is in the way that the *APSS* uses heuristics based on desired end-states to guide the creation of genomes in the genetic algorithm.

Several planning systems use hierarchies of software agents to perform particular functions tailored to the purposes of that system. For example AFS/HAC coordinates the activities of multiple agents through a supervenient architecture in which the higher levels provide goals to be executed by the lower levels, and knowledge about outcomes is passed upwards [161]. By comparison, the agents in the *APSS* system are tailored to

particular functions (execution monitoring, re-planning) and are centrally controlled by a single master agent (planning executive).

So far, there has only been one system built to apply simulations to the mission operational environment. The OpSim system validated the utility of simulations in determining the significance of differences between the observed state of execution and the plan [14]. One of the major requirements identified in the OpSim research was that the planning support system should be extended to not only find the significant differences between the execution and the plan, but also take action to 'fix' the plan. The *APSS* system has been specifically designed to meet that requirement.

D. Variable Parameters

Many of the activities performed in the *Anticipatory Planning Support System* can be modified or 'tuned' by changing parameters. To simplify the experimental process the *APSS* prototype includes dialogs that allow interactive changes of the parameters. This allows the tester to vary as few or as many parameters as desired and observe the effects within the system. The following sections describe the variable parameters for the entities in the system and the system components themselves.

1. Entity Parameters

As was previously discussed, TacEntities have many attributes. Among these are movement rates, effective ranges, and maximum strength values. If desired, these variables can be changed by the tester. The attrition model uses an exponential function to determine losses over time. The parameters used in the attrition model can be

modified through a dialog box.. The HexCells also have attributes, such as the 'go' ability, the amount of buildup, the amount of vegetation, etc. The effect of the different terrain modifies on movement can be adjusted.

2. StateDifferenceAnalyzer Parameters

The StateDifferenceAnalyzer determines the likelihood that a *Planned State* can be reached from the *Actual State*. There are two adjustable parameters used by this analyzer. They are the importance (or weight) given to the distance between the two states of each TacEntity, and the importance given to the disparity in strengths of each TacEntity. The system includes the ability to select from three different kinds of difference functions.

3. Re-planning Priority Parameters

The re-planning priority is a function of three factors: the likelihood that the node will occur, the temporal proximity of the *Node* to the *Actual State*, and the number of *Branches* already planned from that *Node*. Each of these has a weight parameter associated with them, and the weights are adjustable through a dialog. Several different re-planning priority functions were used in the experiments.

4. End-States and Viability

When building the plan, the user determines the desired end-state for the friendly and enemy forces. A desired end-state is composed of six components. The first is that force's objective, and is measured by the distance of its TacEntities from that objective. The second is its strength, which the force wants to maximize. The third is the no-later-than completion time, which should not be passed. The fourth is the opposing force's

objective, which their TacEntities should be kept away from. The fifth is the opposing force's strength, which should be minimized. The sixth and final component is the opposing force's no-later-than time, by which its objective should not be met. Both the friendly and the enemy force have a desired end-state.

These components taken together, and given the appropriate weights, allow the planner to specify overall goals. For instance, a heavily weighted friendly objective and a lightly weighted friendly end strength represents a "get to the objective at all costs" strategy. Normally, however, the objective and end-strength are more nearly balanced in importance. A higher weight on a short no-later-than time represents a strategy of getting to the objective in a hurry, with less consideration for end-strength, or a 'terrain-oriented' strategy. If higher weights are given to the enemy not accomplishing its objective, end-strength, or time, then this represents a 'force-oriented' strategy.

Given the desired end-states, the viability of a Node for both the friendly and enemy forces can be determined. For example, a Node whose state indicates that the friendly force won't get to the objective on time, or won't have much strength left when the objective is reached will reduce the viability. Of course, this assumes that the end-state is weighted towards getting friendly forces on the objective in time. Note that it is possible for a Node to simultaneously have high or low viability for both forces.

5. Genetic Algorithm Parameters

The genetic algorithm used by the BranchesGenerator has a number of parameters. The number of generations can be adjusted, as can the number of genomes to be produced in each generation. The probability that chromosomes (paths for a

specific TacEntity) will cross over from one new child to the other can be changed. Similarly, the probability that a new child will contain a mutation (a random path for a specific TacEntity) is adjustable.

The fitness function used by the genetic algorithm is adjustable through two parameters. The fitness function uses the friendly and enemy viabilities of each newly generated genome to determine its fitness. The two viabilities can be assigned a weight ranging from –1.0 to 1.0, and the fitness function attempts to maximize the sum of the two weighted viabilities. To get an idea of how this works, consider some of the extreme cases.

If the friendly viability has a weight of 1.0 and the enemy viability a weight of –1.0 the genomes with the highest probability of reproduction are those where the friendly force is accomplishing its mission and the enemy force is not. Swapping the two values results in the opposite situation. If both viabilities have a weight of 1.0, then the most-fit genomes are those where both the friendly and enemy force are achieving their goals.

E. Test Situations

The fundamental idea behind the Anticipatory Planning process is to expend more planning effort ahead of the most likely paths of the actual operation, and less ahead of the least likely paths. For the methodology employed in the *Anticipatory Planning Support System* to be considered successful, there are three situations that it must properly handle. First, if the actual path of the operation is already represented in the PlanDescription, then planning should occur primarily ahead of that path. Second, if

the actual path of the operation deviates from a path within the PlanDescription, but then returns to a path in the PlanDescription, the planning should stay ahead of the deviation, then focus ahead of the re-converged path. Third, if the actual path is not represented in the PlanDescription, then planning should stay primarily in front of the actual path. The tests for each situation are described below, along with analysis of the results of the tests. For each test, a simple, medium, and complex scenario is used.

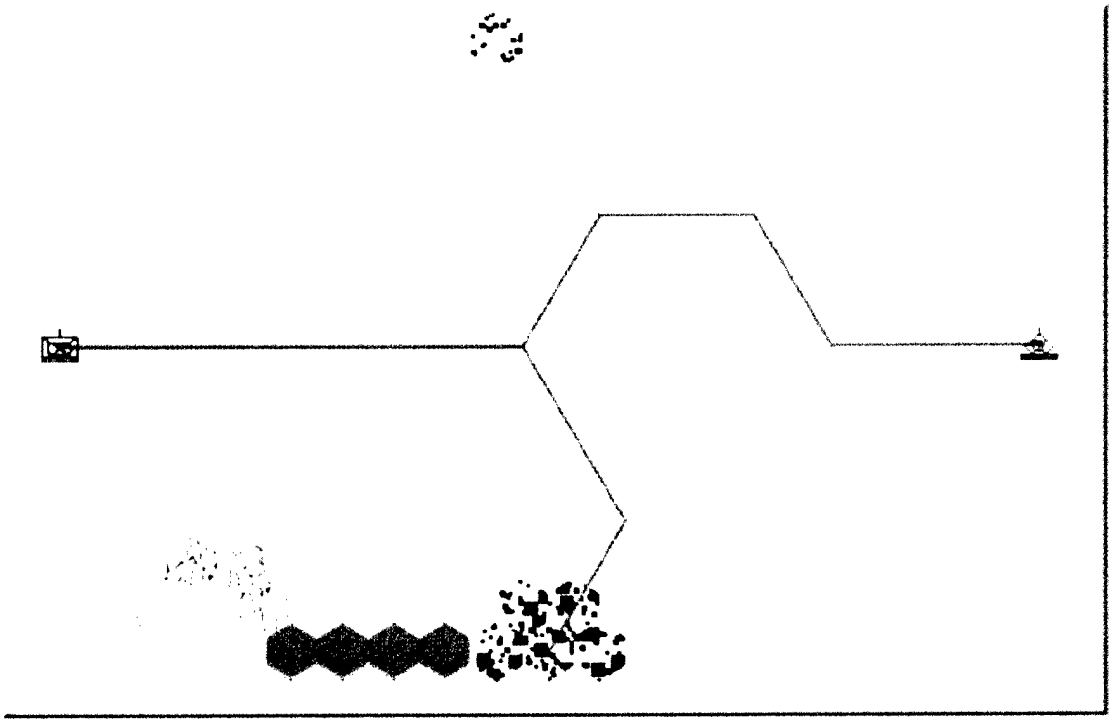

Figure 12: Simple Scenario

F. Test Scenarios

In the simple scenario a single friendly TacEntity and a single enemy TacEntity are placed on opposite ends of the HexGrid. Each has an objective on the other side of

the HexGrid from their start point. The Branches in the PlanDescription bring the opponents into contact in the middle, and allow them to bypass each other in several ways. See Figure 12 for the start state of the simple scenario.

In the medium scenario a battalion of four friendly companies starts on the west end of the HexGrid, with an objective on the east end. A battalion of three enemy companies starts on the east end, with an objective on the west. The Branches keep the enemy companies together as they advance. The friendly companies start out together, but are allowed to disperse in some branches. See Figure 13 for the start state of the medium scenario.

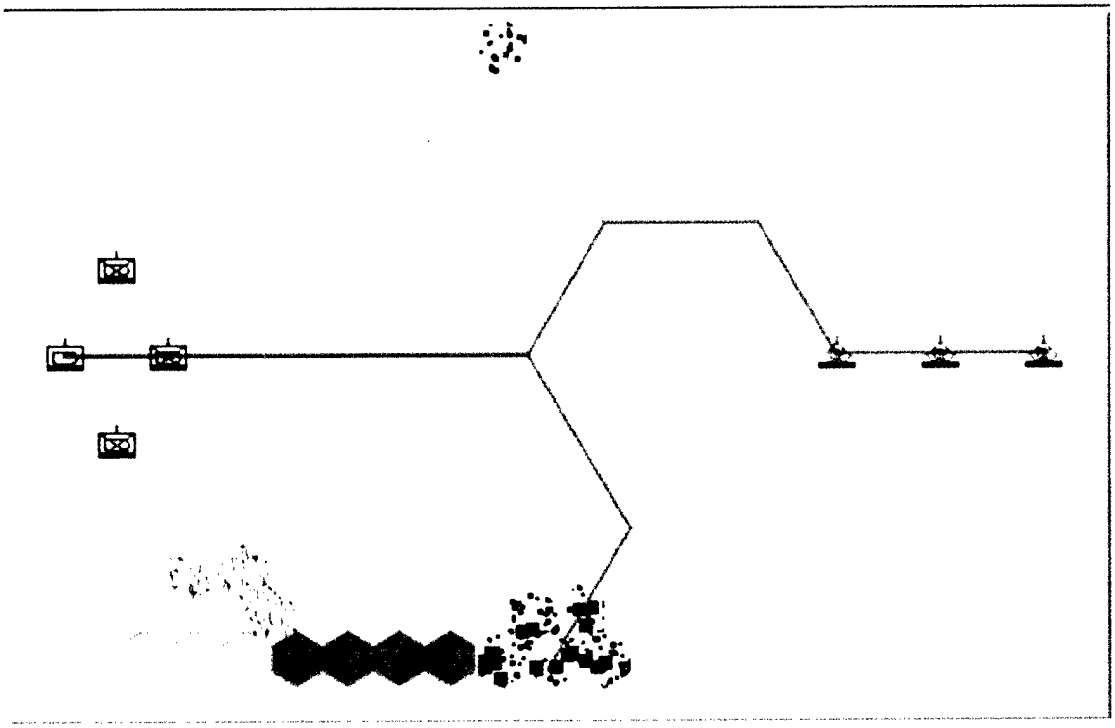

Figure 13: Medium Scenario

In the complex scenario, the friendly and enemy forces start with two battalions. Again, their objectives are on the other side of the HexGrid. Several Branches bring the opponents into direct contact. Several others allow them to maneuver around each other. See Figure 14 for the start state of the complex scenario.

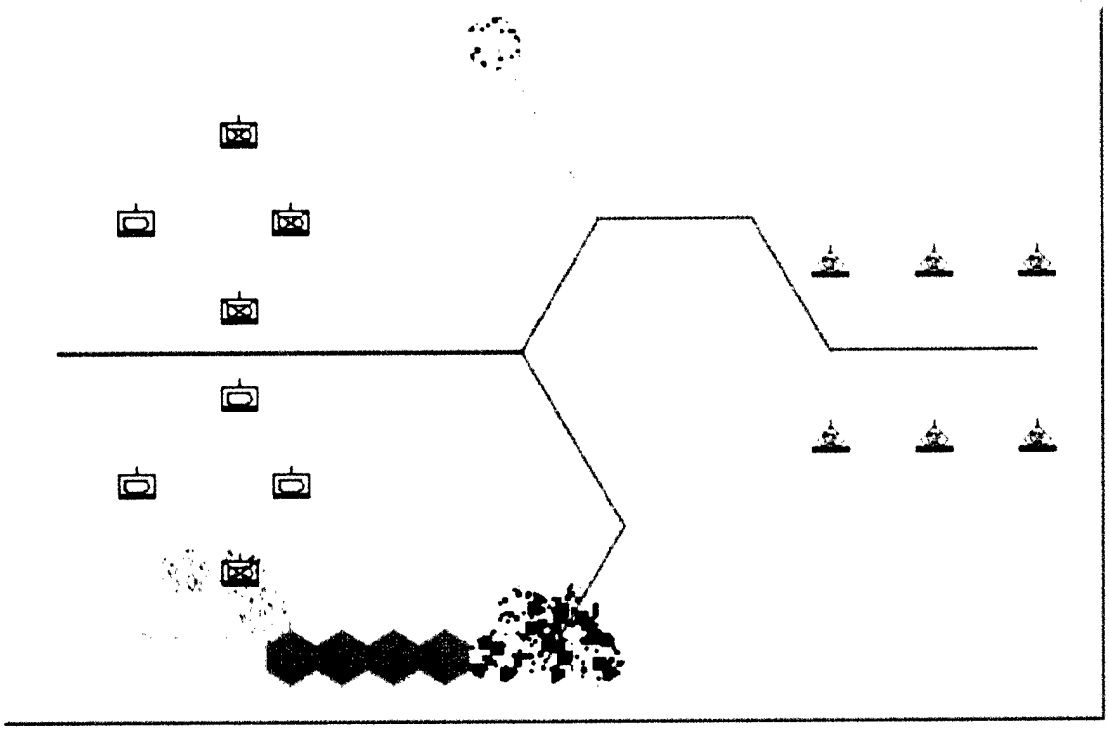

Figure 14: Complex Scenario

G. Test Conditions

The three situations described above were tested using the simple, medium, and complex scenario. The results and analysis of the nine combinations are described below. For clarity, a path through the plan, or the operation, consisting of a sequence of

Branches will be referred to as a 'plan path.' The path taken by a TacEntity across the HexGrid will be referred to as a 'ground path.'

In all of these tests constraints have been placed on the system to maintain clarity in the depiction of the plan description. The PlanningExecutive has been limited to placing Planners on no more than five Nodes. This allows its priority mechanism to be observed. Each of the Planners is limited to creating three new Branches. One Branch runs the BranchesGenerator with a fitness function that favors friendly viability of the end Node. The second Branch is the result of a fitness function that favors enemy viability. The third Branch comes from a fitness function that favors the maximization of both friendly and enemy viabilities. This is similar to the often used "best-case, worst-case, medium-case" approach to testing.

Note that the term 'favors' does not imply that one side is using a good strategy and the other a bad strategy. The genetic algorithm is always following the strategies indicated in the desired end states for both opponents in creating new generations. After the candidate branches have been created in each generation, the fitness function causes the 'favored' ones to be more likely to reproduce in the next generation.

Table 5: Color Representations in Plan Description

Color Text	Meaning	Color	Greyscale
Black	Very Low		Black
Red	Low		Dark Grey
Yellow	Medium		Light Grey
Green	High		Medium Grey

H. Explanation of Figures

Several figures showing PlanDescriptions are used throughout this chapter to depict the function of the *APSS*. For discussion purposes, the start Node of the Plan Description is at level one, and the levels increase from left to right. Also, time advances from left to right, and the placement of the Nodes in the X-axis are proportional to their time stamp against the maximum time represented.

There are four colors used to represent likelihood and viability. See Table 5 for the description and meanings of the colors. Branches are colored to represent the likelihood that the end state of each Branch will be achieved. Nodes are colored to indicate the viability of that state from the friendly perspective. The colors are still distinguishable in greyscale images.

The Test Executive provides periodic Actual State updates. Normally, the system always plans ahead of the Actual State updates by placing one Planner on the Node holding the update. This makes sense, since the Actual State update Node has the highest likelihood. For testing purposes, planning ahead of the Actual State updates has been inhibited for clarity.

When Nodes in the PlanDescription have been bypassed in time, their likelihood is set to zero, making them turn black. This not only represents the inability to achieve that Node anymore, it helps depict the progress of time through the PlanDescription. The Branches may change colors several times as the likelihood of their end Nodes changes, but once their end Node is passed, the Branches retain the color representing the last likelihood of that Node. This has the beneficial effect of leaving the most

closely followed plan paths composed of green Branches, making it easy to determine if the *APSS* is correctly assessing their likelihoods.

Somewhere in the PlanDescription there is always a Node with a small purple dot in it, or a Branch with a purple outline. This merely indicates the currently selected Node or Branch. The full *APSS* screen has a preview panel that displays the state at a selected Node, or the state at a given time in a selected Branch.

When ExecutionMonitors have been assigned to a Node, the Node is outlined in blue. Planners assigned to a Node cause it to be outlined in purple. In those instances where a Node is simultaneously hosting an ExecutionMonitor and a Planner, the Node is outlined in orange. When the Node is not hosting any agents, it is outlined in black. In greyscale images it is difficult to distinguish the black outlines from the blue (black and very dark grey), and the orange outlines from the purple (both light grey). However, most of the discussion will center on where the Planners are, and the light grey outlines always indicate Planners.

I. Analysis of Test Situations

This section contains descriptions and depictions of the actual tests run to confirm that the *APSS* properly handles the three test situations.

1. Actual Path is Already Represented

When the actual plan path of the operation is already represented in the PlanDescription, the majority of the re-planning effort should stay ahead of that plan path. The likelihood of each Node on the plan path should remain high throughout the

operation. The likelihood of the remaining Nodes should diminish as the operation progresses.

The screen capture in Figure 15 shows the state of the *APSS* at seventeen minutes into the operation. The Test Executive is providing Actual State updates along the plan path represented by the first (top) Branch out of each Node. The four Nodes in the lower-left corner represent the four Actual State updates received by the system. In this case, planning ahead of the Actual State updates has been suppressed.

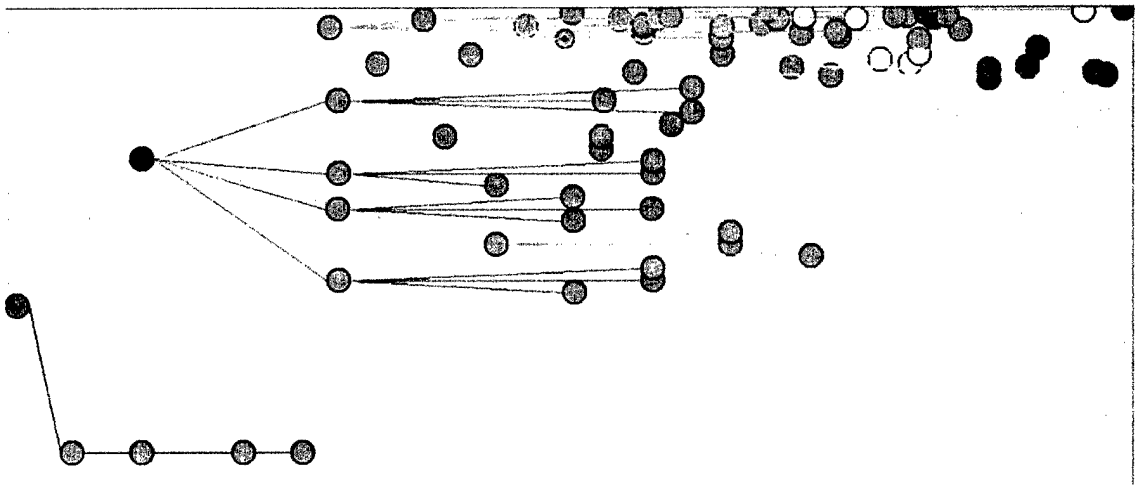

Figure 15: Simple Scenario – 17 Minutes

At this point all but two of the level three nodes have had their likelihood rated as yellow or red. The two green plan paths are the actual plan path and a plan path that the Planner created after a previous update. The previous efforts of the Planners along the less promising paths is discernable. Notice the Nodes at level three that have exactly three children. Now that the actual state update is approaching the actual plan path Node

at level three, the PlanningExecutive has shifted all of the Planners to paths in front of that Node, where the higher priorities for planning occur.

By the time the Test Executive has posted an actual state update at the twenty-five minute mark (see Figure 16), the *APSS* has marked only the actual plan path as very likely, one path as medium, and all the others as unlikely. All of the Planners are now working on Nodes in advance of the actual plan path. No new plan paths have been generated on the less likely Nodes. The *APSS* correctly follows the actual plan path to the conclusion of the operation.

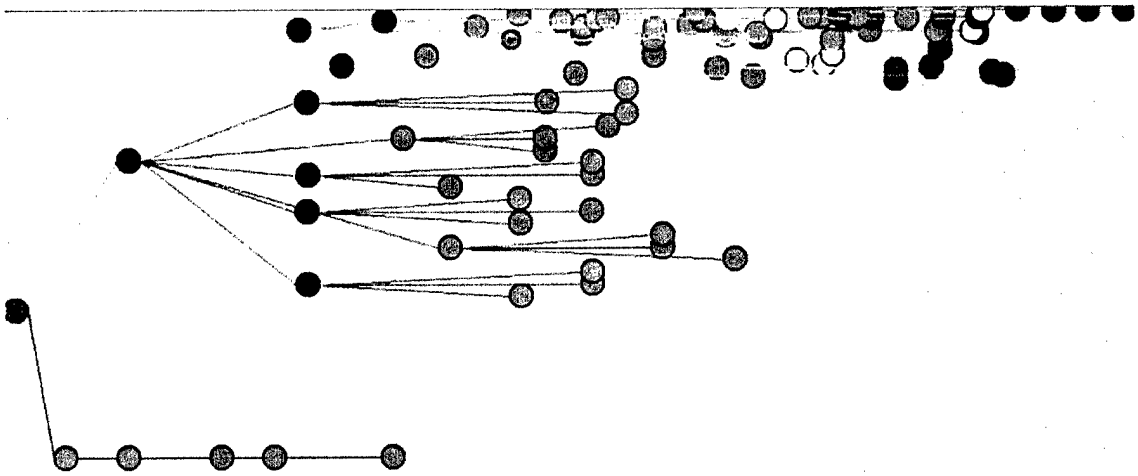

Figure 16: Simple Scenario - 25 Minutes

In the simple scenario, the re-planning effort remains in front of the actual plan path through the PlanDescription. Although the system 'hedged its bets' and planned ahead of several initially likely paths, by seventeen minutes into the operation it had

clearly isolated the actual plan path. An examination of the likelihoods associated with each Node reveals that the differences in the planned states and the Actual States are very sharp. This makes sense in light of the small number of TacEntities involved and the very different initial plan paths.

The *APSS* performed substantially the same when processing the medium scenario. It did take a little bit longer to focus on the given path, but did prioritize planning ahead of the actual plan path throughout the operation. This makes sense because with more TacEntities in play the differences between planned and actual states is not quite as clear cut. For example, one of the Nodes had a high likelihood overall, even though in the actual state one of the TacEntities was nowhere near its planned position.

In processing the medium scenario, the PlanningExecutive tended to place the Planners that were not ahead of the actual plan path on Nodes ahead of likely paths that had been created by previous planners. This also makes sense when you consider the original plan paths are deliberately designed to be distinct from each other. As the Planners create new plan paths they are effectively 'filling in the gap' between the original plan paths.

The *APSS* performed substantially the same way when processing the complex scenario. Note that the complexity of the scenario is not related to the complexity of the plan. The plans for all three of the scenarios had roughly the same depth and branching factor. The slight delay in isolating the actual plan path was evident here, just as in the medium scenario. However, the system tended to place even more planning priority

along the actual plan path when the actual state was very close in time to a planned Node in the actual plan path. This suggests that having plan paths composed of many shorter Branches is better for operation of the *APSS* than plan paths composed of fewer long Branches.

2. Actual Path Diverges and Converges

This section describes the test of the situation where the actual plan path diverges from a planned path, then converges back onto it. Each of the three scenarios was copied and an additional path added. This new path included the major deviation, and concluded in a state as similar to an existing planned state as possible. For these scenarios, planning ahead of the Actual State updates was turned back on.

For the simple scenario, the *APSS* performed as expected. The more Actual State updates that were received indicating the operation was on a completely different path, the less planning occurred ahead of the original plan path in the PlanDescription. Also, more planning occurred ahead of the Actual State updates. Once the Actual State updates approached the planned state in the existing Node, more planning occurred ahead of the original plan path.

It is important to note what happens with the planning effort when the deviation converges back to the original plan path. At this point, both the Node representing the Actual State and the original Node are assigned almost equal likelihood measures by the ExecutionMonitors. Also, the following planned Nodes and any new Nodes created ahead of the Actual State updates will have very high likelihood measures. The result is that planning effort is divided between the two paths. Although this might seem a

duplication of effort, functionally this is no different than if the two plan paths actually merged. The PlanDescription is being modified to allow the substantially equivalent existing Node to replace the Actual State update Node. This will make the plan description look more like a directed acyclic graph than a tree.

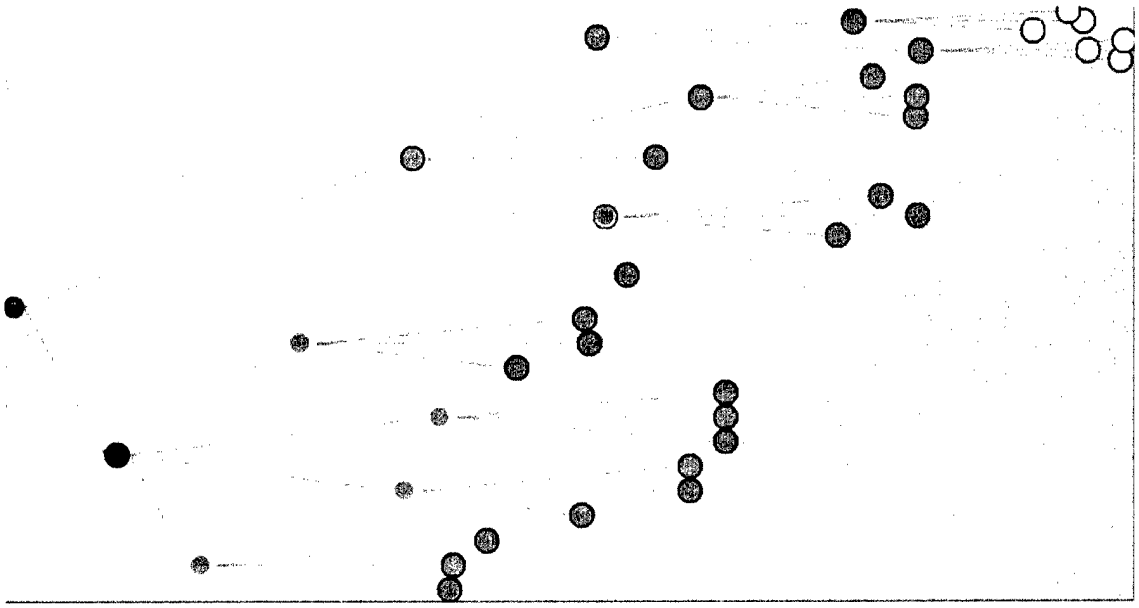

Figure 17: Medium Scenario - 12 Minutes

For the medium scenario, more planning occurred ahead of the deviation as it progressed. The screen capture in Figure 17 shows the *APSS* display at twelve minutes into the operation. The first Actual State update occurred at about seven minutes. It is apparent that at this point the divergence was not too significant, since four of the five Planners were assigned ahead of the existing path (in the top half of the display). The remaining planner worked on the Node holding the Actual State. The second update, at twelve minutes, clearly shows that the divergence has been detected. Only one Planner

is working ahead of the existing plan path. The other four are planning ahead of the Actual State.

From the twelve minute update all the way through the twenty-seven minute update (when the Actual State is very close to converging on the existing plan path) all five Planners stay ahead of the Actual State. At the twenty-seven minute update, two of the five Planners once again start planning ahead of the existing plan path. See Figure 18 for the screen capture of the twenty-seven minute update.

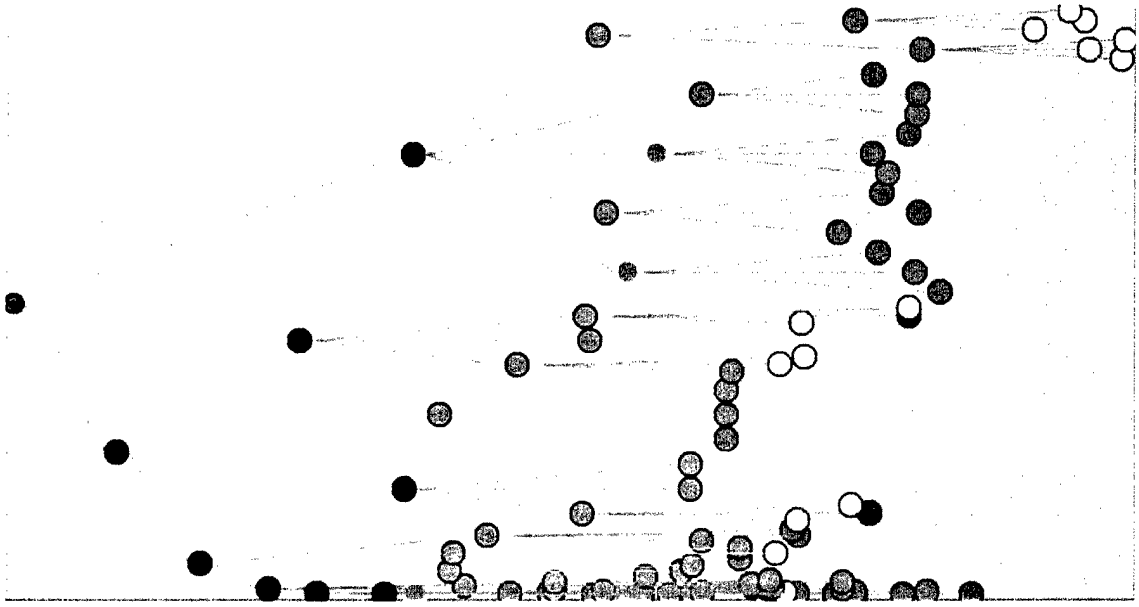

Figure 18: Medium Scenario - 27 Minutes

From this point on the duplicated effort discovered in the simple scenario is apparent. Throughout the remainder of the operation, the Planning Executive divides the planning effort between the existing plan path that accurately tracks the operation, and the Actual State updates. The duplicated effort is identifiable in Figure 19, a screen

capture at fifty-six minutes into the operation. The most likely paths are the existing plan path that best represents the actual operation (towards the top) and the plan path of the Actual State Updates. The planning effort along these two plan paths is clearly discernable from the number of planned Nodes along those paths.

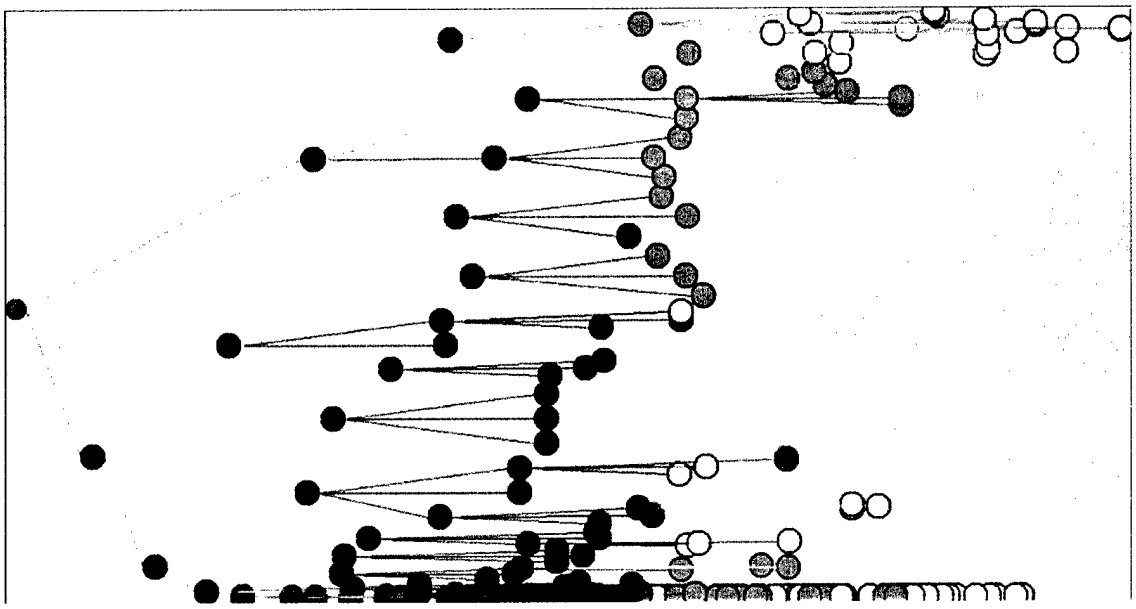

Figure 19: Medium Scenario - 56 Minutes

The complex scenario provided substantially similar results. The planning effort followed the divergence and then returned to the existing plan path that best represented the actual operation. In both the medium and complex scenario, the PlanningExecutive did not return the planning effort to the existing plan path until the Actual State update was within three or four minutes of the Node representing the convergence point. Although this would seem to suggest that the system is not identifying the convergence fast enough, it is important to remember the planning that has been performed ahead of

the Actual State updates. As those updates converge on the existing Node, that planning effort is producing new Nodes that are just as likely as the Nodes ahead of the convergence Node. Although the shift of planning effort does not occur until slightly before the convergence Node, plenty of useful planning has been performed before the convergence occurs. However, this highlights the need to treat the PlanDescription as a directed graph and allow the representation of the plan to actually converge.

3. Actual Path Completely Diverges

The final series of experiments examines the conduct of re-planning when the actual state updates from the operation completely diverge from any of the existing plan paths. For this series, planning ahead of Actual State updates has been turned on. The first experiment in this series used the complex scenario with a very straight-forward PlanDescription containing only a few plan paths. These plan paths deliberately made all activities occur in the north (top) of the HexGrid. The control PlanDescription used by the simulator contained an additional plan path that took the activity to the south. Similar experiments were run on the simple and medium scenarios with completely similar results. The complex scenario will serve to illustrate the experimental series.

The first Actual State update occurred at approximately five minutes into the operation and caused four of the five planners to plan ahead of the existing plan path. The remaining planner created three new Branches from the Node of the Actual State update. This was expected, since the difference in positions and enemy strength was not too great. In other words, although the enemy forces had started to move south, they could still quite easily move back to the north, restoring the existing plan path.

The second Actual State update occurred at about nine minutes, and the Execution Monitors detected significant differences. This time, the Planning Executive assigned four of the Planners ahead of the Actual State, and only one Planner along the existing plan path.

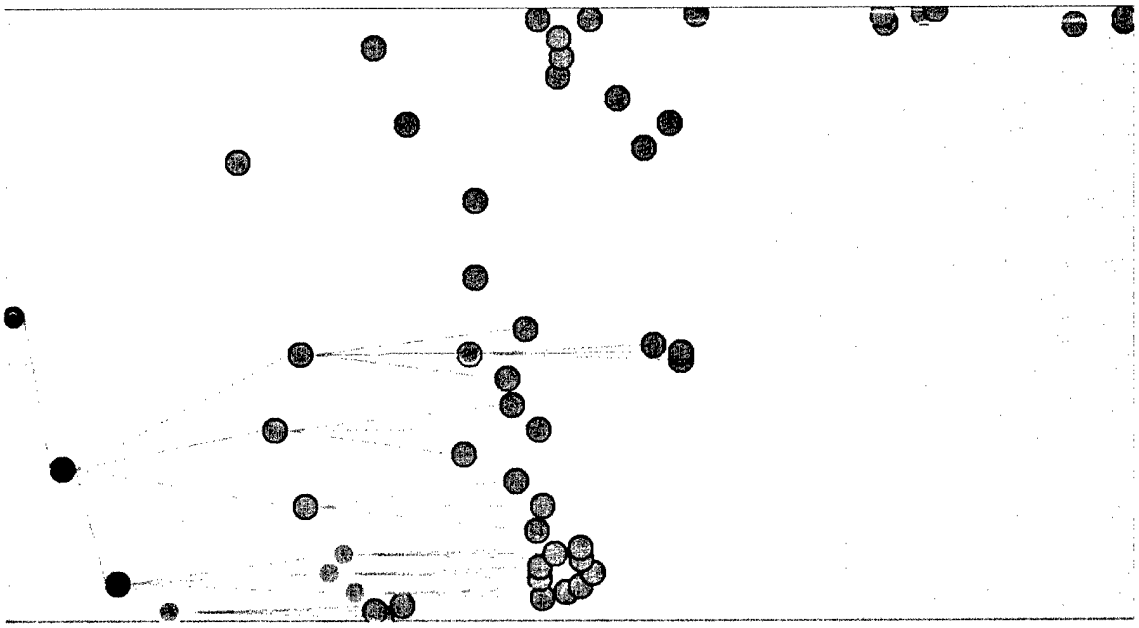

Figure 20: Complex Scenario - 14 Minutes

With the arrival of the third Actual State update at fourteen minutes into the operation the Execution Monitors rated all Branches along the existing plan path as 'less likely' and the Planning Executive assigned all five Planners on the newly created plan paths ahead of the Actual State Node (see Figure 20 for the screen capture). This is exactly the desired behavior of the system. That is, when the situation has altered

drastically from any of the existing plan paths, the Planning Executive must focus the planning effort on developing new plan paths ahead of the Actual State of the operation.

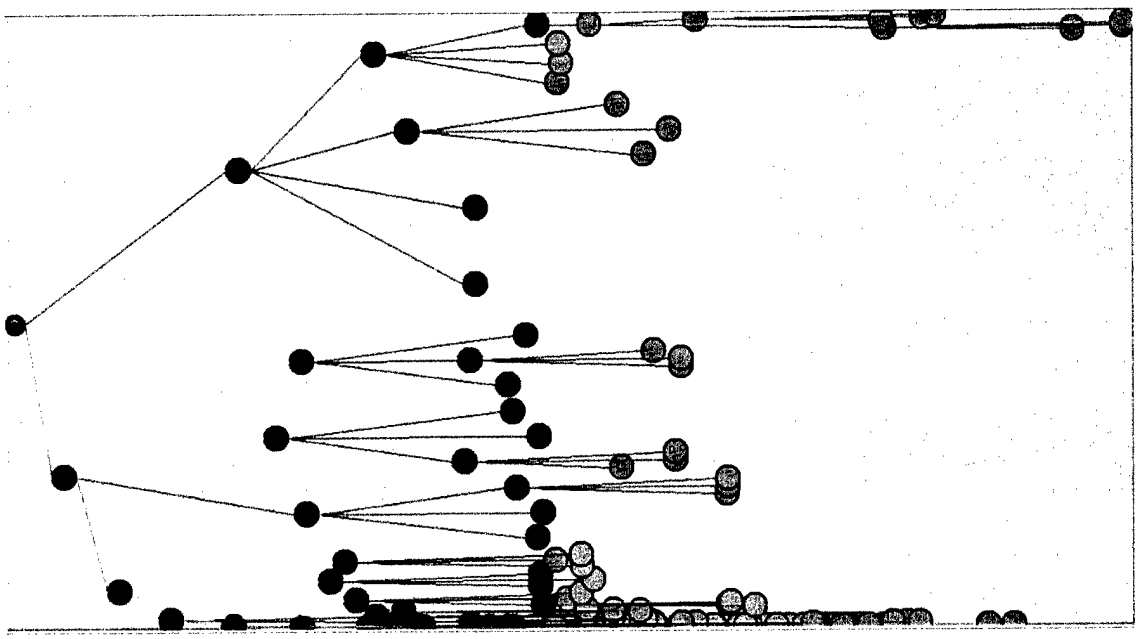

Figure 21: Complex Scenario - 48 Minutes

For the next thirty-four minutes the additional Actual State updates only serve to confirm that the operation remains diverged from the existing plan paths. Throughout this period almost all of the existing plan paths remain 'unlikely' and none of them receive any planning effort. The screen capture at Figure 21 shows all of those plan paths colored red (dark gray, in grayscale). It also shows the large cluster of newly created plan paths on the bottom of the screen, ahead of the Actual State update Nodes.

For clarity, that portion of the PlanDescription has been expanded in Figure 22 to show what is happening to nodes at the tenth level and beyond. The screen capture

shows one Planner at work ahead of the most recent update Node, three ahead of the immediately previous update Node, and one Planner working on a Node created only two iterations prior.

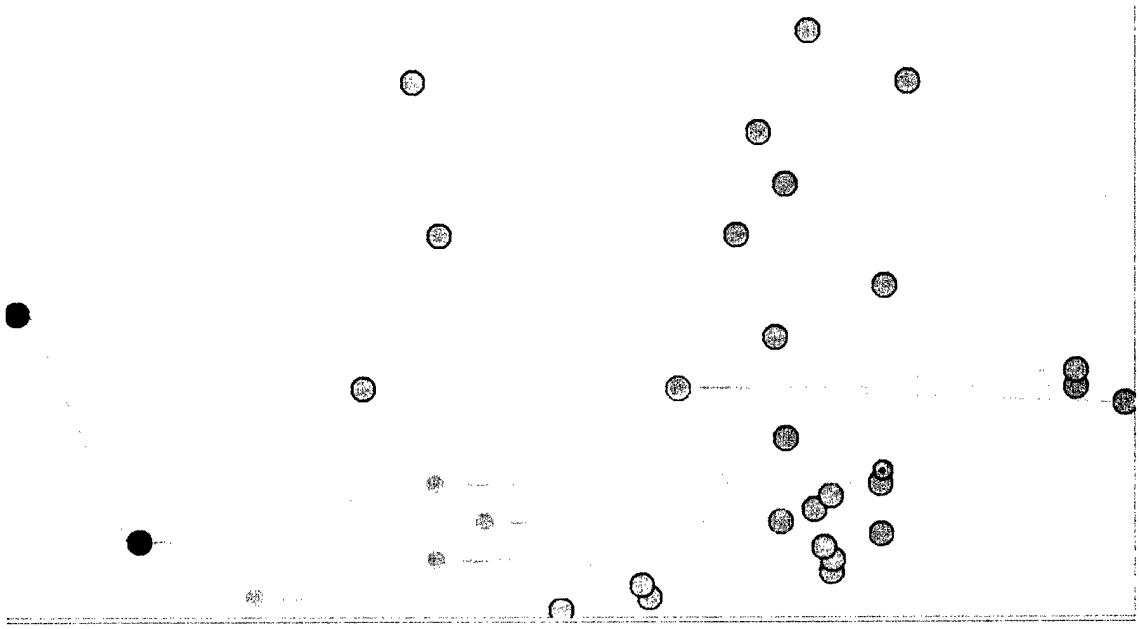

Figure 22: Complex Scenario - 48 Minutes (expanded)

CHAPTER VI

SUMMARY AND CONCLUSIONS

A. Summary

The purpose of the Anticipatory Planning methodology is to develop, monitor, and create as many viable options for the commander as possible, and to constrain the planning effort along the most fruitful plan paths. A great deal of work has been done in the fields of planning under uncertainty, artificial intelligence, and simulation (Chapter II describes this previous work). Only recently have these tools been focused on helping human military planners manage and take advantage of the vast amount of battlefield information. Modern computing technologies such as software agents, genetic algorithms, and operationally-focused simulation can be applied to provide the necessary capabilities for implementing the methodology. Chapter III describes the design that integrates these technologies into a system that enables the Anticipatory Planning Methodology to be realized. The implementation of that design, discussed in Chapter IV, has produced a prototype *Anticipatory Planning Support System.* This prototype was used to conduct a series of experiments, described in Chapter V, to determine the validity of the methodology. The conclusions drawn from those experiments are presented in the next section.

This research is not intended to produce a fully autonomous planning system. Human military planners do not really want a system to do all of the planning for them. They want a system that supports their planning by taking over the mundane tasks,

manage information, keeping track of possibilities, and helping them determine whether a plan is viable and when it is in danger of failure. The *Anticipatory Planning Support System* promises to provide those desired capabilities, and is designed to help commanders and their staffs see the possible flows of the battle so they can take actions early enough to influence the outcome.

B. Conclusions

The overview at the end of Chapter I provides a list of the objectives of this research. All of those research objectives have been met. A common Plan Description has been created that works correctly in all parts of the system. The three software agents that detect plan deviations, prioritize re-planning, and produce new plan paths have been developed. The Anticipatory Planning methodology has been implemented in a prototype system that enables useful experimentation. An unexpected but very useful additional contribution was discovered in the ability to use the *APSS* as a means of stimulating planning systems.

1. PlanDescription

The common *Plan Description* works effectively in every part of the system: plan building and task assignment, plan visualization, simulation to determine interactions, and testing. The experiments identified the need to convert the tree-like Plan Description into a directed graph allowing convergence of plan paths to eliminate duplication of effort. The Plan Description is being modified to allow replacement of

the Actual State Node with the substantially similar existing Node. This is a connection issue; the internal workings of the Nodes and Branches will not require modification.

2. Agents

The three agents that do all the work inside the *APSS* performed as expected. The Execution Monitors successfully determined differences between an Actual State and a planned state, and their use of the State Difference Analyzer created correct estimates of likelihood. The Planners used a genetic algorithm to rapidly produce a number of new possibilities from a given node. These new Branches were created with an awareness of the desired end-states of the opponents. The Planning Executive successfully processed recommendations from Execution Monitors and assigned Planners to Nodes in accordance with a priority scheme that ensured the planning effort occurred along the most fruitful plan paths.

3. Anticipatory Planning Methodology

The methodology performs the required actions as designed. The system successfully followed actual planning paths. It also noted and planned ahead of divergences from existing plan paths. If the actual plan path representing the operation re-converged on an existing planned path, planning effort also returned to the existing plan path. There was some duplication of effort in this case. The cause and the proposed solution have already been noted.

4. Prototype System

The *APSS* prototype facilitates the planning process. The human planner can task units, observe interactions, build plans, and consider "what-if scenarios." At any

time, even when the system is active, the commander or staff select Branches, see a preview, and run a simulation to observe the interaction of the entities. They can also continue to create new Branches and plan paths.

5. Means of Stimulating Planning Systems

Finally, the *APSS* is able to serve as a *Stimulator* of planning systems. Although this was not an objective of this research, it proved to be a valuable tool in testing the prototype and the methodology. Since the *Stimulator* produces Actual States of the operation at a given time stamp it should prove able to stimulate other planning and simulation systems. The only requirement would be a conversion of the Actual State into whatever representation is used in the target system.

C. Future Work

Now that the prototype system has been developed and the initial proof of concept completed there are many directions this research can take. Although not exhaustive, the following sections describe additional work that can be performed starting from this foundation.

1. Eliminate duplicate TacEntity Paths

One of the goals of this research was efficient implementation of its constituent parts. The PlanDescription could benefit from a modification that eliminates duplicate plan paths by the same TacEntity. For example, if a Tank Company takes exactly the same actions in two different Branches, then there are two exact copies of the TacEntityTaskList for that company. Some creative mapping of Branches to

TacEntityTaskLists would eliminate this duplication. If a new Branch was created that desired a modification to that common TacEntityTaskList, then a new copy of the list could be made and modified, leaving the original list unchanged in all of the existing Branches that rely on it.

2. End-States in a Course of Action Analysis Tool

There is an interesting way to use the desired end-states that bears investigation. It may well be possible for a new agent (call it an analyzer) to change the desired end-states, then run simulations to determine the outcomes. The analyzer could compare the outcomes to the Actual State updates and determine which strategy the enemy is using. It may also be useful to expand how the end states are designed to allow a more flexible representation of different strategies.

3. Converge Actual-Plan Path with Existing Plan Paths

The biggest issue identified in the experiments was the duplication of effort that occurred when the Actual State updates were already represented in existing plan paths. The solution, re-implementation of the PlanDescription as a directed graph, has already been discussed.

4. Connect *APSS* to Existing Military Systems

Ultimately, a more robust version of *APSS* would have to receive its Actual State from the Common Operational Picture produced by battlefield information systems. Also, integration of the *APSS* with existing military simulations, such as OneSAF, would be useful.

REFERENCES

[1] H. Wass de Czege, Jr., Personal Communication (regarding Anticipatory Planning), October, 1999.

[2] U. S. Army, *Field Manual 101-5: Staff Organization and Operations*, Washington, D.C.: U.S. Government Printing Office, 1997.

[3] D. J. Ragsdale, Personal Communication (regarding Information Technology Operations Center), April, 2000.

[4] U. S. Army, *FM 100-5: Operations*, Washington, D.C.: U.S. Government Printing Office, 1993.

[5] U. W. Pooch and J. A. Wall, *Discrete Event Simulation: A Practical Approach*, Boca Raton, Florida: CRC Press, 1993.

[6] J. Banks and J. S. Carson, *Discrete-Event System Simulation*, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1984.

[7] J. R. Surdu, "Connecting Simulation to the Mission Operational Environment," Ph.D. Dissertation, Texas A&M University, College Station, Texas, 2000.

[8] A. M. Law and W. D. Kelton, *Simulation Modeling & Analysis*, New York: McGraw-Hill, Inc., 1991.

[9] R. J. Hillestad and L. Moore, *The Theater-Level Campaign Model: A Research Prototype for a New Generation of Analysis Model*, Santa Monica, California: RAND Corporation, 1996.

[10] D. I. A. Cohen, *Introduction to Computer Theory*, New York: John Wiley & Sons, 1991.

[11] T. H. Naylor and J. M. Finger, "Verification of Computer Simulation Models," *Management Science*, vol. 14, pp. 92-101, 1967.

[12] J. Banks, J. S. Carson, II, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation*, Upper Saddle River, New Jersey: Prentice-Hall, 2001.

[13] B. J. Bortscheller and E. T. Saulnier, "Model Reusability in a Graphical Simulation Package," in *Proceedings of the Winter Simulation Conference*, Arlington, Virginia, December 13-16, 1992, pp. 764-772.

[14] J. R. Surdu and U. W. Pooch, "Simulation Technologies in the Mission Operational Environment," *Simulation*, vol. 74, no. 3, pp. 138-160, March, 2000.

[15] B. P. Zeigler, "Review of Theory in Model Abstraction," in *Proceedings of the SPIE Annual International Symposium on Aerospace / Defense Sensing, Simulation, and Controls (AeroSense): Enabling Technology for Simulation Science II*, Orlando, Florida, April 13-17, 1998, pp. 2-12.

[16] A. F. Sisti and S. D. Farr, "Modeling and Simulation Enabling Technologies for Military Applications," in *Proceedings of the 1996 Winter Simulation Conference*, Coronado, California, 8-11 December, 1996, pp. 877-883.

[17] G. Adkins and U. W. Pooch, "Computer Simulations: A Tutorial," *Computer*, vol. 10, no. 4, pp. 12-17, April, 1977.

[18] J. Banks, "Introduction to Simulation," in *Proceedings of the 1999 Winter Simulation Conference*, Phoenix, Arizona, December 5-8, 1999.

[19] F. P. Hoeber, *Military Applications of Modeling: Selected Case Studies*, New York: Gordon and Beach Science Publishers, 1982.

[20] S. Vincent, "Input Data Analysis," in *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice,* J. Banks, ed., New York: John Wiley & Sons, 1998, pp. 55-92.

[21] W. J. Davis, "On-Line Simulation: Need and Evolving Research Requirements," in *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice,* J. Banks, ed., New York: John Wiley and Sons, Inc., 1998, pp. 465-516.

[22] W. J. Davis, X. Chen, A. Brook, and F. A. Awad, "Implementing On-line Simulation with the World Wide Web," *Simulation*, vol. 73, no. 1, pp. 40-53, January, 1998.

[23] W. J. Davis, "On-Line Simulation for Torpedo Avoidance," Available at http://www-msl.ge.uiuc.edu/~brook/boat, November, 1998.

[24] S. D. Anderson, "Issues in Interleaved Planning and Execution," in *Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments, AAAI Technical Report WS-98-02,* R. Bergmann and A. Kott, ed., Madison, Wisconsin: AAAI Press, 1998, pp. 62-66.

[25] B. Ferren, "Some Brief Observations on the Future of Army Simulation," *Army RD&A*, vol. 99, no. 3, pp. 31-37, March, 1999.

[26] R. McNab and F. W. Howell, "Using Java for Discrete Event Simulation," in *Proceedings of the Twelfth UK Computer and Telecommunications Performance Engineering Workshop (UKPEW)*, University of Edinburgh, Edinburgh, Scotland, September 29 - October 3, 1996, pp. 219-228.

[27] K. A. DeJong, "An Analysis of Behavior of a Class of Genetic Adaptive Systems," Ph.D. Dissertation, University of Michigan, Ann Arbor, Michigan, 1975.

[28] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, Massachusetts: Addison Wesley, 1989.

[29] J. Giarratano and G. Riley, *Expert Systems Principles and Programming*, Boston: PWS-Kent Publishing Company, 1989.

[30] C. L. Forgy, "RETE: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem," *Artificial Intelligence*, vol. 19, no. 1, pp. 17-37, September, 1982.

[31] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Englewood Cliffs, New Jersey: Prentice-Hall, 1995.

[32] P. O. Doyle, "Search Methods," Available at http://www-cs-students.stanford.edu/~pdoyle/quail/notes/pdoyle/search.html, April 21, 2000.

[33] P. Pantel, "Intelligent Adversary Searches," Unpublished Paper, December, 1997.

[34] A. Newell, J. C. Shaw, and H. A. Simon, "Chess Playing Programs and the Problem of Complexity," *IBM Journal of Research and Development*, vol. 2, pp. 320-335, October, 1958.

[35] N. J. Nilsson, *Problem-Solving Methods in Artificial Intelligence*, New York: McGraw-Hill, 1971.

[36] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210-229, April, 1959.

[37] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers II - Recent Progress," *IBM Journal of Research and Development*, vol. 11, no. 6, pp. 601-617, November, 1967.

[38] T. P. Hart and D. J. Edwards, "The Tree Prune (TP) Algorithm," Technical Artificial Intelligence Project Memo 30, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1961.

[39] A. L. Brudno, "Bounds and Valuations for Shortening the Scanning of Variations," *Problemy Kibernetiki (Problems of Cybernetics)*, vol. 10, pp. 141-150, May, 1963.

[40] J. R. Slagle, "Game Trees, *m* & *n* Minimaxing, and the *m* & *n* Alpha-Beta Procedure," Artificial Intelligence Group Report 3, University of California, Lawrence Radiation Laboratory, Livermore, California, 1963.

[41] J. R. Slagle and J. K. Dixon, "Experiments with Some Programs that Search Game Trees," *Journal of the Association for Computing Machinery*, vol. 16, no. 2, pp. 189-207, April, 1969.

[42] A. Kotok, "A Chess Playing Program for the IBM 7090," AI Project Memo 41, MIT Computation Center, Cambridge, Massachusetts, 1962.

[43] R. D. Greenblatt, D. E. Eastlake, and S. D. Crocker, "The Greenblatt Chess Program," in *Proceedings of the 1967 AFIPS Fall Joint Computer Conference*, Anaheim, California, November 14-16, 1967, pp. 801-810.

[44] D. E. Knuth and R. W. Moore, "An Analysis of Alpha-Beta Pruning," *Artificial Intelligence*, vol. 6, no. 4, pp. 293-326, December, 1975.

[45] J. Pearl, "The Solution for the Branching Factor of the Alpha-Beta Pruning Algorithm and its Optimality," *Communications of the Association for Computing Machinery*, vol. 25, no. 8, pp. 559-564, August, 1982.

[46] H. Berliner, "The B*-Tree Search - A Best-First Proof Procedure," *Artificial Intelligence*, vol. 12, no. 1, pp. 23-40, May, 1979.

[47] A. J. Palay, "The B*-Tree Search - New results," *Artificial Intelligence*, vol. 19, no. 2, pp. 145-163, October, 1982.

[48] G. C. Stockman, "A MiniMax Algorithm Better than Alpha-Beta?," *Artificial Intelligence*, vol. 12, no. 2, pp. 179-196, August, 1979.

[49] P. O. Doyle, "Planning," Available at http://www-cs-students.stanford.edu/~pdoyle/quail/notes/pdoyle/planning.html, April 30, 2000.

[50] A. Tate, "A Review of AI Planning Technologies," in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate, ed., San Mateo, California: Morgan Kaufmann, 1990, pp. 26-49.

[51] A. Newell and H. A. Simon, "GPS, a Program that Simulates Human Thought," in *Lernende Automaten,* H. Billing, ed., Munich, Germany: R. Oldenbourg, 1961, pp. 109-124.

[52] E. Charniak and D. McDermott, *Introduction to Artificial Intelligence*, Reading, Massachusetts: Addison-Wesley, 1985.

[53] R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, vol. 2, no. 3-4, pp. 189-208, December, 1971.

[54] R. E. Fikes, P. E. Hart, and N. J. Nilsson, "Learning and Executing Generalized Robot Plans," *Artificial Intelligence*, vol. 3, no. 1-3, pp. 251-288, January, 1972.

[55] E. D. Sacerdoti, "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence*, vol. 5, no. 2, pp. 115-135, June, 1974.

[56] C. A. Knoblock, "An Analysis of ABSTRIPS," in *Proceedings of the First International Conference on Artificial Intelligence Planning Systems (AIPS92)*, College Park, Maryland, June 15-19, 1992, pp. 126-135.

[57] E. D. Sacerdoti, "The Nonlinear Nature of Plans," in *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI 75)*, Tbilisi, USSR, September 3-8, 1975, pp. 206-214.

[58] A. Tate, "Generating Project Networks," in *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, Cambridge, Massachusetts, August 22-25, 1977, pp. 888-893.

[59] M. Stefik, "Planning With Constraints (MOLGEN: Part 1)," *Artificial Intelligence*, vol. 16, no. 2, pp. 111-140, May, 1981.

[60] D. Wilkins, "Domain-Independent Planning: Representation and Plan Generation," *Artificial Intelligence*, vol. 22, no. 3, pp. 269-301, April, 1984.

[61] D. Wilkins, *Practical Planning: Extending the Classical AI Planning Paradigm*, San Mateo, California: Morgan Kaufmann, 1988.

[62] D. Chapman, "Planning for Conjunctive Goals," Technical Report TR-802, MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts, November, 1985.

[63] Q. Yang, "A Theory of Conflict Resolution in Planning," *Artificial Intelligence*, vol. 58, no. 1-3, pp. 361-392, December, 1992.

[64] L. Pryor and G. Collins, "Planning for Contingencies: A Decision-Based Approach," *Artificial Intelligence Research*, vol. 4, no. 1, pp. 287-339, May, 1996.

[65] G. J. Sussman, "A Computational Model of Skill Acquisition," Technical Report AI-TR-297, MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts, August, 1973.

[66] D. Warren, "WARPLAN: A System for Generating Plans," Technical Report 76, University of Edinburgh, Edinburgh, Scotland, May, 1974.

[67] D. Warren, "Generating Conditional Plans and Programs," in *Proceedings of the Summer Conference on Artificial Intelligence and the Simulation of Behavior (AISB-76)*, University of Edinburgh, Edinburgh, Scotland, July, 1976, pp. 344-354.

[68] M. A. Peot and D. E. Smith, "Conditional Nonlinear Planning," in *Proceedings of the First International Conference on Artificial Intelligence Planning Systems (AIPS-92)*, College Park, Maryland, June 15-19, 1992, pp. 189-197.

[69] D. McDermott, "Planning and Acting," *Cognitive Science*, vol. 2, no. 2, pp. 71-109, April-June, 1978.

[70] S. A. Vere, "Planning in Time: Windows and Durations for Activities and Goals," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 5, no. 3, pp. 246-267, May, 1983.

[71] M. P. Georgeff and A. L. Lansky, "Reactive Reasoning and Planning," in *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, Washington, July 13-17, 1987, pp. 677-682.

[72] D. Chapman and P. Agre, "Abstract Reasoning as Emergent from Concrete Activity," in *Reasoning About Actions & Plans - Proceedings of the 1986 Workshop,* M. P. Georgeff and A. L. Lansky, ed., San Mateo, California: Morgan Kaufmann, 1986, pp. 411-424.

[73] P. Agre and D. Chapman, "PENGI: An implementation of a Theory of Activity," in *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, Seattle, Washington, July 13-17, 1987, pp. 268-272.

[74] M. Wooldridge and N. Jennings, "Intelligent Agents: Theory and Practice," *Knowledge Engineering Review*, vol. 10, no. 2, pp. 115-152, June, 1995.

[75] M. J. Schoppers, "Universal Plans for Reactive Robots in Unpredictable Environments," in *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August, 1987, pp. 1039-1046.

[76] J. Ambros-Ingerson and S. Steel, "Integrating Planning, Execution, and Monitoring," in *Proceedings of the Seventh National Conference on Artificial Intelligence*, Saint Paul, Minnesota, August 21-26, 1988, pp. 83-88.

[77] K. Golden, O. Etziano, and D. Weld, "Omnipotence Without Omniscience: Efficient Sensor Management for Planning," in *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, Washington, August 1-4, 1994, pp. 1048-1054.

[78] C. Knoblock, "Planning, Executing, Sensing, and Replanning for Information Gathering," in *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Quebec, Canada, August 19-25, 1995, pp. 1686-1693.

[79] E. P. D. Pednault, "ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus," in *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Ontario, Canada, May 15-18, 1989, pp. 324-332.

[80] J. Reye, "Reusability via Formal Modeling," in *Proceedings of the ITS'96 Workshop on Architectures and Methods for Designing Cost-Effective and Reusable ITSs*, Montreal, Canada, June 10, 1996, pp. 1-5.

[81] M. Drummond, "Situated Control Rules," in *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Ontario, Canada, May 15-18, 1989, pp. 103-113.

[82] D. Olawsky and M. Gini, "Deferred Planning and Sensor Use," in *Proceedings of the 1990 DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, San Diego, California, November 5-8, 1990, pp. 166-174.

[83] A. Tate, "Coordinating the Activities of a Planner and an Execution Agent," in *Proceedings of the Second NASA Conference on Space Telerobotics*, Pasadena, California, January 31 - February 2, 1989, pp. 385-393.

[84] K. Currie and A. Tate, "O-Plan: the Open Planning Architecture," *Artificial Intelligence*, vol. 52, no. 1, pp. 49-86, November, 1991.

[85] Artificial Intelligence Applications Institute, "O-Plan - Open Planning Architecture," Available at http://www.aiai.ed.ac.uk/~oplan/, April 27, 2000.

[86] A. Barrett, S. Soderland, and D. S. Weld, "Effect of Step-Order Representations on Planning," Technical Report 91-05-06, Department of Computer Science and Engineering, University of Washington, Seattle, Washington, 1991.

[87] D. McAllester and D. Rosenblitt, "Systematic Non-Linear Planning," in *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, California, July 14-19, 1991, pp. 634-639.

[88] D. McDermott, "Regression Planning," *International Journal of Intelligent Systems*, vol. 6, no. 4, pp. 357-416, 1991.

[89] J. S. Penberthy and D. Weld, "UCPOP: A Sound, Complete, Partial Order Planner for ADL," in *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, Cambridge, Massachusetts, October 26-29, 1992, pp. 103-114.

[90] O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson, "An Approach to Planning with Incomplete Information," in *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, Cambridge, Massachusetts, October 26-29, 1992, pp. 11-125.

[91] J. G. Carbonell, J. Blythe, O. Etzioni, Y. Gil, R. Joseph, D. Kahn, C. Knoblock, S. Minton, A. Perez, S. Reilly, M. Veloso, and X. Wang, "PRODIGY 4.0: The Manual and Tutorial," Technical Report CMU-CS-92-150, Department of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, June, 1992.

[92] E. Fink and M. Veloso, "PRODIGY Planning Algorithm," Technical Report CMU-CS-94-123, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, March, 1994.

[93] P. Stone and M. Veloso, "User-Guided Interleaving of Planning and Execution," in *New Directions in AI Planning,* M. Ghallab and A. Milani, ed., Amsterdam: IOS Press, 1996, pp. 103-112.

[94] N. Carver and V. Lesser, "A Planner for the Control of Problem Solving Systems," *IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Planning, Scheduling and Control*, vol. 23, no. 6, pp. 1519-1536, November - December, 1993.

[95] E. A. Hansen, "Cost-Effective Sensing During Plan Execution," in *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Seattle, Washington, August 1-4, 1994, pp. 1029-1035.

[96] J. S. Penberthy and D. S. Weld, "Temporal Planning with Continuous Change," in *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI '94)*, Seattle, Washington, August 1-4, 1994, pp. 1010-1015.

[97] P. Haddawy and M. Suwandi, "Decision-Theoretic Refinement Planning using Inheritance Abstraction," in *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, Chicago, Illinois, June 13-15, 1994, pp. 266-271.

[98] P. R. Cohen, M. S. Atkin, and E. A. Hansen, "The Interval Reduction Strategy for Monitoring Cupcake Problems," in *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, Brighton, United Kingdom, August 8-12, 1994.

[99] S. J. Ceci and U. Bronfenbrenner, ""Don't Forget to take the Cupcakes out of the Oven": Prospective Memory, Strategic Time-Monitoring, and Context," *Child Development*, vol. 56, pp. 152-164, February, 1985.

[100] N. Kushmerick, S. Hanks, and D. Weld, "An Algorithm for Probabilistic Planning," *Artificial Intelligence*, vol. 76, no. 1-2, pp. 239-286, September, 1995.

[101] D. Draper, S. Hanks, and D. Weld, "A Probabilistic Model of Action for Least-Commitment Planning with Information Gathering," in *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, Seattle, Washington, July 29-31, 1994, pp. 178-186.

[102] D. Draper, S. Hanks, and D. Weld, "Probabilistic Planning with Information Gathering and Contingent Execution," in *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, Chicago, Illinois, June 13-15, 1994, pp. 31-36.

[103] R. P. Goldman and M. S. Boddy, "Conditional Linear Planning," in *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, Chicago, Illinois, June 13-15, 1994, pp. 80-85.

[104] R. P. Goldman and M. S. Boddy, "Representing Uncertainty in Simple Planners," in *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning*, Bonn, Germany, May 24-27, 1994, pp. 238-245.

[105] R. St. Amant, Y. Kuwata, and P. R. Cohen, "Monitoring Progress with Dynamic Programming Envelopes," in *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, Washington, D.C., November 5-8, 1995, pp. 426-433.

[106] A. L. Lansky, "Action-Based Planning," in *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, University of Chicago, Chicago, Illinois, June, 1994, pp. 110-115.

[107] D. Joslin and M. E. Pollack, "Passive and Active Decision Postponement in Plan Generation," in *Proceedings of the Third European Workshop on Planning (EWSP'95)*, Assisi, Italy, September 27-29, 1995.

[108] A. Tate, "Representing Plans as a Set of Constraints - the <I-N-OVA> Model," in *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, Edinburgh, Scotland, May 29-31, 1996, pp. 221-228.

[109] G. Collins and L. Pryor, "Planning Under Uncertainty: Some Key Issues," in *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Quebec, Canada, August 20-25, 1995, pp. 1567-1573.

[110] W. J. Partridge, "Automating the U.S. Army's Planning and Decision-Making Process for the Conduct of Military Operations," Available at http://carlisle-www.army.mil/usacsl/divisions/std/branches/keg/97TermII/partridg.htm, January 10th, 2000.

[111] K. Kelly, D. Gardner, D. Thompson, and J. Wynn, "Tactical Decision-Making Seizes Initiative," *The Edge: The MITRE Advanced Technology Newsletter*, vol. 3, no. 2, pp. 6-7, June, 1999.

[112] L. Seligman, P. Lehner, C. Elsaesser, K. Smith, and D. Mattox, "Information Monitoring for Decision Support," *The Edge: The MITRE Advanced Technology Newsletter*, vol. 3, no. 2, pp. 8-9, 11, June, 1999.

[113] L. Seligman, P. Lehner, K. Smith, C. Elsaesser, and D. Mattox, "Decision-Centric Information Monitoring," Unpublished Paper, June, 1999.

[114] J. J. Wynn, J. D. Roberts, and M. O. Kinkead, "Visualizing the Wargame - Web-based Applications for Viewing a Course of Action (COA)," in *Proceedings of the 1999 International Conference on Web-Based Modeling and Simulation*, San Francisco, California, January 17-20, 1999, pp. 227-323.

[115] P. Delaney, Personal Communication (regarding Technology Voids), October, 1999.

[116] P. Emmerman, Personal Communication (regarding Research into Course of Action Analysis and Comparison), January 5, 2000.

[117] Beckman Institute for Advanced Science and Technology, "Interactive Displays Federated Laboratory," Available at http://www.ifp.uiuc.edu/IDFL/, January 19th, 2000.

[118] J. E. Kirzl, "Command and Control Evaluation in the Information Age," in *Proceedings of the RTO Meeting 38 (RTO-MP-38 AC/323(SAS) TP/12), Studies, Analysis and Simulation (SAS) Panel, 1999 Symposium on Modeling and Analysis of Command and Control*, Issy les Moulineaux, France, June, 1999, pp. 3-1 to 3-13.

[119] A. Tolk, "Requirements for Simulation Systems when being used as Decision Support Systems," in *Proceedings of the 1999 Simulation Interoperability Workshop*, Orlando, Florida, September 12-17, 1999.

[120] Department of Defense, "DoD Modeling and Simulation (M&S) Master Plan," Available at http://www.dmso.mil/documents/policy/msmp/index.html, March 7th, 2000.

[121] W. H. Lunceford, Jr., G. Schow, R. Richardson, and R. Alexander, "Results of the DARPA Course of Action Analysis (COAA) Proof of Principle Pilot Test," in *Proceedings of the 1999 Spring Simulation Interoperability Workshop*, Orlando, Florida, March 14-19, 1999.

[122] D. Alberts, "The Future of Command and Control with DBK," in *Dominant Battlespace Knowledge,* S. E. Johnson and M. C. Libicki, ed., Washington, D.C.: National Defense University Press, 1996, pp. 67-88.

[123] K. Brandt, "Modeling and Simulation Links with Command & Control (C2) Systems," in *Proceedings of the 1999 Fall Simulation Interoperability Workshop*, Orlando, Florida, September 12-17, 1999.

[124] D. Kang and R. J. Roland, "Military Simulation," in *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice,* J. Banks, ed., New York: John Wiley & Sons, 1998, pp. 645-658.

[125] S. Rodio, "ModSAF," Available at http://www.stricom.army.mil/STRICOM/E-DIR/ES/MODSAF, March, 1999.

[126] J. Balash, "Janus," Available at http://www.stricom/army.mil/PRODUCTS/ JANUS, March, 1999.

[127] M. Rogers, "Warfighter's Simulation," Available at http://www.stricom.army. mil/PRODUCTS/WARSIM, March, 1999.

[128] M. Loper and S. Seidensticker, *The DIS Vision: A Map to the Future of Distributed Simulation*, Orlando, Florida: Institute for Simulation and Training, Central Florida University, 1994.

[129] J. R. Surdu, G. Haines, and U. W. Pooch, "OpSim: a Purpose-built Distributed Simulation for the Mission Operational Environment," in *Proceedings of the International Conference on Web-Based Modeling and Simulation (WebSim 1999)*, San Francisco, California, 17-20 January, 1999, pp. 69-74.

[130] C. L. Blais and W. M. Garrabrants, "Simulation in Support of Mission Planning," in *Proceedings of the Advanced Simulation Technologies Conference (ASTC 1999): Military, Government, and Aerospace (MGA) Simulation Symposium*, San Diego, California, April 11-15, 1999, pp. 117-122.

[131] J. J. Lee and P. A. Fishwick, "Simulation-Based Planning for Computer Generated Forces," *Simulation*, vol. 63, no. 5, pp. 299-315, November, 1994.

[132] J. J. Lee, "A Simulation-Based Approach for Decision Making and Route Planning," Ph.D. Dissertation, University of Florida, Gainesville, Florida, 1996.

[133] J. J. Lee and P. A. Fishwick, "Simulation Based Planning in Support of Multi-Agent Scenarios," Technical Report 97-001, Computer and Information Science and Engineering Department, University of Florida, Gainesville, Florida, 1997.

[134] S. D. Anderson and P. R. Cohen, "On-Line Planning Simulation," in *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*, Edinburgh, Scotland, May 29-31, 1996, pp. 3-10.

[135] P. A. Fishwick, G. Kim, and J. J. Lee, "Improved Decision Making Through Simulation Based Planning," *Simulation*, vol. 67, no. 5, pp. 315-327, November, 1996.

[136] S. Barone and J. D. Roberts, "Uses of Simulation for Military Planning," in *Proceedings of the 1998 Fall Simulation Interoperability Workshop*, Orlando, Florida, September 14-18, 1998.

[137] J. Sheehan, L. Obermeyer, and M. Hopkins, "Order of Battle Data Interchange Format and Access Tool," in *Proceedings of the 1998 Fall Simulation Interoperability Workshop*, Orlando, Florida, September 14-18, 1998.

[138] K. S. Collier, "Automated Decision Support Systems Enabled by Models and Simulations -- A "Leap-ahead" Technology Recommendation for the US Army After Next Time Frame (2020-2025)," in *Proceedings of the Advanced Simulation Technologies Conference (ASTC 1999): Military, Government and*

Aerospace (MGA) Simulation Symposium, San Diego, California, April 11-15, 1999, pp. 3-8.

[139] S. Franklin and A. Graesser, "Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents," in *Intelligent Agents III: Proceedings of the Workshop on Agent Theories, Architectures, and Languages,* M. J. W. a. N. R. J. Jörg P. Müller, ed., Berlin: Springer-Verslag, 1997, pp. 21-36.

[140] M. Lejter and T. Dean, "A Framework for the Development of Multi-Agent Architectures," *IEEE Expert*, vol. 11, no. 6, pp. 47-59, December, 1996.

[141] L. Spector, "Supervenience in Dynamic-World Planning," Technical Report PhD 92-5, The Institute for Systems Research, University of Maryland, College Park, Maryland, 1992.

[142] J.-P. Bouche, J.-P. Floch, and M. Michel, "Using Command Agents for Military Planning," in *Proceedings of the 1999 Spring Simulation Interoperability Workshop*, Orlando, Florida, March 14-19, 1999.

[143] P. R. Cohen, M. L. Greenberg, D. M. Hart, and A. E. Howe, "Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments," *AI Magazine*, vol. 10, no. 3, pp. 32-48, Fall, 1989.

[144] D. M. Hart, S. D. Anderson, and P. R. Cohen, "Envelopes as a Vehicle for Improving the Efficiency of Plan Execution," in *Proceedings of the 1990 DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, San Diego, California, November 5-9, 1990, pp. 71-76.

[145] Experimental Knowledge Systems Laboratory, "Phoenix: An Adaptable Planner for a Complex Real-time Environment," Available at http://eksl-www.cs.umass.edu/research/phoenix.html, April 27, 2000.

[146] C. Applegate, C. Elsaesser, and J. Sanborn, "An Architecture for Adversarial Planning," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no. 1, pp. 186-194, January/February, 1990.

[147] D. Wilkins and K. L. Myers, "Integrating Planning and Reactive Control," in *Proceedings of the Third International Symposium on Artificial Intelligence, Robotics, and Automation for Space*, New Orleans, Louisiana, November 4-6, 1994, pp. 161-164.

[148] D. E. Wilkins and K. L. Myers, "A Common Knowledge Representation for Plan Generation and Reactive Execution," *Journal of Logic and Computation*, vol. 5, no. 6, pp. 731-761, December, 1995.

[149] J. B. Gilmer, Jr. and F. J. Sullivan, "Combat Simulation Trajectory Management," in *Proceedings of the 1996 Simulation Multiconference: Military, Government, and Aerospace Simulation*, New Orleans, Louisiana, April 8-11, 1996, pp. 236-241.

[150] J. B. Gilmer, Jr., "Alternative Implementations of Multitrajectory Simulation," in *Proceedings of the 1998 Winter Simulation Conference*, Washington, D.C., December 13-16, 1998, pp. 865-872.

[151] J. B. Gilmer, Jr. and F. J. Sullivan, "Multitrajectory Simulation Performance for Varying Scenario Sizes," in *Proceedings of the 1999 Winter Simulation Conference*, Phoenix, Arizona, December 5-8, 1999, pp. 1137-1146.

[152] S. Al-Hassan, J. B. Gilmer, Jr., and F. J. Sullivan, "A Simulation State Management Technique Sensitive to Measures of Effectiveness," in *Proceedings of the 1997 SCS Simulation MultiConference: Military, Government, and Aerospace Simulation*, Atlanta, Georgia, April 6-10, 1997, pp. 95-100.

[153] J. B. Gilmer and F. J. Sullivan, "Recursive Simulation to Aid Models of Decisionmaking," in *Proceedings of the Winter Simulation Conference (WSC 2000)*, Orlando, Florida, December 10-13, 2000, pp. 958-963.

[154] C. Elsaessar, "Adversarial Planner," Available at http://www-leav.army.mil/nsc/warsim/reason/adplan/getstart/eagle/index.htm, March 21st, 2000.

[155] C. Elsaessar, "Adversarial Planner User Guide," Available at http://www-leav.army.mil/nsc/warsim/reason/adplan/index.htm, March 21st, 2000.

[156] C. C. Hayes and J. L. Schlabach, "FOX-GA: A Planning Support Tool for assisting Military Planners in a Dynamic and Uncertain Environment," in *Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments, Technical Report WS-98-02,* R. Bergmann and A. Kott, ed., Madison, Wisconsin: AAAI Press, 1998, pp. 21-26.

[157] J. L. Schlabach, C. C. Hayes, and D. E. Goldberg, "FOX-GA: A Genetic Algorithm for Generating and Analyzing Battlefield Courses of Action," *Evolutionary Computation*, vol. 7, no. 1, pp. 45-68, Spring, 1999.

[158] R. Slife, Personal Communication (regarding Research into Course of Action Analysis and Comparison), January 5th, 2000.

[159] DARPA, "Command Post of the Future (CPOF) Project," Available at http://dtsn.darpa.mil/iso/, January 27, 2001.

[160] R. H. Kewley, "Automated Tactical Course of Action Development," Technical Report, Operations Research Center, United States Military Academy, West Point, New York, September, 1999.

[161] M. Atkin, D. L. Westbrook, and P. R. Cohen, "Domain-General Simulation and Planning with Physical Schemas," in *Proceedings of the Winter Simulation Conference 2000*, Orlando, Florida, December 10-13, 2000, pp. 1730-1738.

[162] M. Atkin, D. L. Westbrook, and P. R. Cohen, "AFS and HAC: Domain-General Agent Simulation and Control," in *Proceedings of the AAAI Workshop on Software Tools for Developing Agents (AAAI-98)*, March 22-24, 1999, 1998, pp. 89-95.

[163] M. Atkin, G. W. King, D. L. Westbrook, B. Heeringa, A. Hannon, and P. R. Cohen, "SPT: Hierarchical Agent Control: A Framework for Defining Agent Behavior (to appear)," in *Proceedings of the Fifth International Conference on Autonomous Agents*, Montreal, Canada, May 28-June 1, 2001.

[164] M. Atkin, D. L. Westbrook, and P. R. Cohen, "Capture the Flag: Military Simulation Meets Computer Games," in *Proceedings of the AAAI Spring Symposium on AI and Computer Games*, Stanford University, Palo Alto, California, March 22-24, 1999.

[165] V. W. Porto, M. Hardt, D. B. Fogel, K. Kreutz-Delgado, and L. J. Fogel, "Evolving Tactics using Levels of Intelligence in Computer-Generated Forces," in *Proceedings of the Third SPIE Enabling Technologies for Simulation Science Conference*, Orlando, Florida, April 5-9, 1999, pp. 262-270.

[166] J. R. Surdu and U. W. Pooch, "A Methodology for Using Intelligent Agents to Apply Simulation Technologies to the Mission Operational Environment," in *Proceedings of the Third SPIE Enabling Technologies for Simulation Science Conference*, Orlando, Florida, April 5-9, 1999, pp. 56-64.

[167] D. E. Wilkins and R. V. Desimone, "Applying an AI Planner to Military Operations Planning," in *Intelligent Scheduling,* M. Fox and M. Zweben, ed., San Francisco, California: Morgan Kaufmann, 1994, pp. 685-709.

[168] J. M. D. Hill, M. S. Miller, J. Yen, and U. W. Pooch, "Tactical Event Resolution Using Software Agents, Crisp Rules, and a Genetic Algorithm," in *Proceedings of the Advanced Simulation Technologies Conference (ASTC 2000): Military, Government, and Aerospace (MGA) Simulation Symposium*, Washington, D.C., April 16-20, 2000, pp. 15-21.

[169] A. Tate, J. Levine, P. Jarvis, and J. Dalton, "Using AI Planning Techniques for Army Small Unit Operations," in *Proceedings of the Fifth International*

Conference on Artificial Intelligence Planning and Scheduling Systems (AIPS 2000), Breckenridge, Colorado, April 15-19, 2000.

[170] J. R. Surdu and U. W. Pooch, "A Methodology for Applying Simulation Technologies in the Mission Operational Environment," in *Proceedings of the IEEE Information Technology Conference*, Syracuse, New York, September 1-3, 1998, pp. 45-48.

[171] J. R. Surdu and U. W. Pooch, "Connecting the Operational Environment to Simulation," in *Proceedings of the Advanced Simulation Technology Conference: Military (ASTC 1999), Military, Government, and Aerospace (MGA) Simulation Symposium*, San Diego, California, April 11-14, 1999, pp. 94-99.

[172] J. R. Surdu and U. W. Pooch, "A Dynamic Hierarchy of Rational Agents to Link Simulation to the Operational Environment," in *Proceedings of the Advanced Simulation Technology Conference: Military (ASTC 2000), Military, Government, and Aerospace (MGA) Simulation Symposium*, Washington, D.C., April 11-14, 2000, pp. 94-99.

[173] U. S. Army, "Global Command & Control System - Army," Available at http://160.147.21.82/wsdocs/stccs/gcssa.htm, January 26th, 2000.

[174] U. S. Army, *Staff Leaders Guide for the Army Battle Command System*, Washington, D.C.: U.S. Government Printing Office, 1998.

APPENDIX A

SOURCE CODE

All of the source code for the Anticipatory Planning Support System is provided on the accompanying compact disc (CD). The CD was written as a standard computer data disc, so it should be readable on any computer. All of the code was written in standard Java, compliant with version 2.0 of that language (Java Developers Kit 1.2). The code was produced using the Borland JBuilder 3.0 development environment, but is readable in any text editor or Java development environment.

VITA

John Mitchell Duval Hill was born in Fort Rucker, Alabama, the second son of an Army couple. He grew up on several Army posts, until his family settled in Austin, Texas, after his father retired from the Army. He attended Andrews Elementary School, Pierce Junior High School, and Anderson High School, graduating as a National Merit Scholar. In 1982, he graduated 24th in his class from the United States Military Academy at West Point, New York, with a bachelor's degree concentrating in Computer Science. He served as a Tank Platoon Leader, Company Executive Officer, and assistant Division Training Officer at Fort Hood, Texas, from 1983-1986. In his next assignment, to Garlstedt, Federal Republic of Germany, from 1986-1990, he served as a Battalion Adjutant, Battalion Maintenance Officer, and Tank Company Commander. From 1990-1992 he earned a master's degree in Computer Science from the University of Texas at Austin. His follow-on assignment from 1992-1995 was teaching at West Point, where he served as an Instructor and then Assistant Professor, and performed duties as the Executive Officer for the Department of Electrical Engineering and Computer Science. He graduated in 1996 from the Command and General Staff College at Fort Leavenworth, Kansas, and then moved to Fort Riley, Kansas, where he served as a Tank Battalion Executive Officer and G-3 Operations Officer. From 1998 to 2001 he attended Texas A&M University, receiving a Ph.D. in Computer Science in May, 2001.

John can most easily be reached through his parents-in-law, 7607 Mesa Drive, Austin, Texas 77831.