CONNECTING SIMULATION TO THE MISSION OPERATIONAL ENVIRONMENT

A Dissertation

by

JOHN R. SURDU

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 2000

Major Subject: Computer Science

CONNECTING SIMULATION TO THE MISSION OPERATIONAL ENVIRONMENT

A Dissertation

by

JOHN R. SURDU

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)	Michael T. Longnecker (Member)
Riccardo Bettati (Member)	Wei Zhao (Head of Department)

May 2000

Major Subject: Computer Science

ABSTRACT

Connecting Simulations to the Mission Operational Environment. (May 2000)

John R. Surdu, B.S., United States Military Academy;

M.B.A., Columbus State University;

M.S., Florida State University

Chair of Advisory Committee: Dr. Udo W. Pooch

A large number of tasks performed by commanders and staffs can be facilitated during operations by the application of simulation technologies. Traditionally the focus of simulation in the Department of Defense (DoD) has been on analysis and training. Simulations designed to facilitate course of action (COA) development and analysis, rehearsal, and operations monitoring can greatly enhance the effectiveness of staffs and commanders. Currently there are no operationally focused simulations built specifically for use during operations.

There are many uses of simulation during operations. Simulations can be used to assist in Course of Action (CAO) development and analysis. They can help orchestrate rehearsals and identify key synchronization issues. Simulations can be used to monitor the course of the operation. Finally, simulations can be used in after action analyses of operations. The research described herein is directly applicable to COA development and analysis, but it concentrates primarily on the use of simulation *during* operations, or *operationally focused simulations*.

Operationally focused simulations are important, because they leverage simulation technology to improve situational awareness, prevent information overload, and help the commander stay inside the enemy's decision cycle. Large Army-wide efforts at improving situational awareness are underway. An operationally focused simulation provides the ability to look at an operation in the present, predict the future, or analyze what has occurred in the past. An operationally focused simulation provides more than just a view of the battle; it facilitates real-time analysis of the implications of friendly and enemy decisions. An operationally focused simulation, like a computer chess analyzer, simulates courses of action into the future and provides information to the commander and staff in a time-efficient manner. This information helps the commander make the right decisions at the right time.

The overall intent of this research is to develop a methodology for using simulations during operations. This methodology will support the construction of tools, which will help decision-makers react quickly and accurately to a rapidly changing operational environment. In support of this research, a prototype simulation and software agent architecture were created. The capabilities of these prototypes as well as experimental results are described.

DEDICATION

This dissertation is dedicated to my wife, Candace, and my children, Thomas and Samantha.

ACKNOWLEDGMENTS

I would like to thank Dr. Udo Pooch for his patience, guidance, and leadership during this research. He has been a wonderful role model, mentor, and guide through this process. He allowed me freedom to research a topic of interest to me and to work at my own pace. More importantly he provide the occasional azimuth checks that kept me focused on the job at hand.

The other members of my committee, Drs. Yen, Bettati, Longnecker, and Hoyle helped make this research stronger. Rather than adversaries, setting themselves as obstacles to be overcome, these gentlemen helped make this research more useful. Any contribution of this work to the field of computer science is due largely to the guidance of these men and Dr. Pooch.

I would like to acknowledge the work and dedication of my wife. She kept the household functioning smoothly. This allowed me to focus on my research without concern for what was happening on the home front.

Finally, I would like to acknowledge the help of the various other graduate students with whom I had frequent interaction and collaboration. Long discussions, class projects, and brain storming sessions with Dan Ragsdale, Mike Miller, John Hill, Curt Carver, and Jeff Humphries have helped build up, tear down, refine, and focus many of the ideas in this research. They were always there when I needed them.

TABLE OF CONTENTS

Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

A. Motivation

A large number of tasks performed by commanders and staffs can be facilitated during operations by the application of simulation technologies. Traditionally the focus of simulation in the Department of Defense (DoD) has been on analysis and training. Simulations designed to facilitate course of action (COA) development and analysis, rehearsal, and operations monitoring can greatly enhance the effectiveness of staffs and commanders. Currently there are no operationally focused simulations, built specifically for use during operations.

The Army Modeling and Simulation Office (AMSO) has identified five modeling and simulation technology voids for the Army After Next[1][1]. This list includes automated decision aids, COA tools, and tactical information aids. The methodology proposed for this research, originally described by Surdu and Pooch [2], intends to address these three technology voids. The Defense Advanced Research Projects Agency (DARPA) has also recognized the importance of simulation in

---

IEEE Transactions on Automatic Control is used as a pattern for format and style.

[1] Army After Next is the vision of the structure and doctrine of the U.S. Army after 2010. It is an ongoing process.

command and control activities. A DARPA concept briefing for the Command Post of the Future (CPoF) project provides a list of several tools that are required to provide input to the *Battlespace Reasoning Manager*, a component of the CPoF. Among these are "Planning and Analysis Applications" and "3D Models and Simulations." In another portion of the briefing, DARPA notes that "Battlespace Reasoning, Analysis, and Simulation" assist the commander's perception and understanding of the battlespace[3].

Clausewitz, a noted military strategist, in *On War (*particularly in his discussion of *friction),* talks about the *feel of the battlefield* and how great commanders have this ability to deal with friction and see through the fog of war [4]. He also notes that this feel of the battlefield only comes with experience. Unfortunately, this experience must be gained through expensive maneuver training exercises and ultimately at the cost of human life. The Army developed a number of facilities (like the Combined Arms Training Centers (CTCs)) and training simulations, such as Corps Battle Simulation (CBS), Brigade and Battalion Battle Simulation (BBS), JANUS (not an acronym), Modular Semi-Automated Forces (ModSAF), Joint Tactical Simulation (JTS), and Warfighter's Simulation (WarSim)[5-7], which attempt to build this experience at relatively low cost. In days of constrained budgets and one-hundred-hour wars, the Army is short of ways of effectively identifying those officers who have this feel of the battlefield. This ability is necessary in every member of the staff in order for the *ad hoc* COA analysis procedure to be effective.

Surdu, Haines, and Pooch[8] discussed many uses of simulation during operations. Simulations can be used to assist in Course of Action (CAO) development

and analysis. They can help orchestrate rehearsals and identify key synchronization issues. Simulations can be used to monitor the course of the operation. Finally, simulations can be used in after action analyses of operations. The research described herein is directly applicable to COA development and analysis, but it concentrates primarily on the use of simulation *during* operations, or *operationally focused simulations*.

Operationally focused simulations are important because they leverage simulation technology to improve situational awareness, prevent information overload, and help the commander stay inside the enemy's decision cycle. Large Army-wide efforts at improving situational awareness are underway [9]. An operationally focused simulation provides the ability to look at an operation in the present, predict the future, or analyze what has occurred in the past. An operationally focused simulation provides more than just a view of the battle; it facilitates real-time analysis of the implications of friendly and enemy decisions. An operationally focused simulation, like a computer chess analyzer, simulates courses of action into the future and provides information to the commander and staff in a time-efficient manner. This information helps the commander make the right decisions at the right time.

B. Research Objectives

The overall intent of this research is to develop a methodology for using simulations *during* operations. This methodology will support the construction of tools,

which will help decision-makers react quickly and accurately to a rapidly changing operational environment. This research includes:

- Research and development of a methodology for using simulations *during* operations, which is based on the use of specially tailored simulations and software agents, decentralized and adaptive control, and information aggregation.
- Development of a methodology for comparing the simulated plan with the real plan that includes:
    - Hierarchical organization of software agents, including:
        - Lower-level, tool agents (many of which are simulated) invoked by analyzer agents, and
        - Higher-level, analyzer agents, which launch other agents as necessary, aggregate information, and make inferences about the progress of the plan.
    - A formal reasoning mechanism to provide for:
        - Dynamic agent control and adaptation,
        - Dynamic adaptation of the simulation, and
        - Synthesis of aggregate information.
- Development of a prototype system to demonstrate the feasibility of this approach.

C. Overview

Chapter II presents a review of the current literature relevant to the domains involved in this research. The purpose of Chapter II is to describe work that has been

done to date and to provide background and motivational material for this dissertation. Domain areas of interest to this dissertation include simulation technology, military art and science, and artificial intelligence (specifically expert systems and software agents).

Chapter III describes the proposed methodology for using simulations during the conduct of operations. This chapter describes the various components of this methodology, paying particular attention to those components that have been prototyped.

Chapter IV gives details of the implemented prototype simulation (OpSim) and the prototype software agents (Operations Monitors, OMs). It describes the interactions of these prototype components and the performance of the overall system.

Chapter V describes the results of this research. It describes the verification and validation procedures used to confirm the correctness and effectiveness of the simulation component. Chapter V also describes some experiments conducted to demonstrate that the software agents perform as designed (verification). Finally Chapter V makes recommendations for future research.

Chapter VI presents the major conclusions of this research and reasserts the contributions of this research to the field.

CHAPTER II

LITERATURE REVIEW

A. Introduction

The research described in this dissertation lies within the intersection of three distinct domains. The first is simulation technology. The second domain is military art and science, particularly the military planning and decision-making process. The third domain is artificial intelligence (AI). From the AI domain, this research makes use of expert system technology and the concept of software agents. Each of these domains will be discussed in sufficient detail to provide background and motivation for this research. In addition, topics related to these domains, such as attrition models and information warfare, will be discussed.

B. Simulation

The purpose of simulation is to provide a tool with which to experiment to "gain some understanding of how a real system behaves" [10] or compare strategies for future operation of the system[11]. Carson and Banks[12] define a simulation as the "imitation of the operation of a real-world process or system over time." Some purposes of simulation are described in Table 1. Since most interesting real-world systems are too

complex to model analytically (i.e., with some closed-form solution), simulation is often used to evaluate a model numerically, gather data, and estimate the true characteristics of the model[10].

As Zeigler noted, “Abstraction is the process underlying model construction whereby a relatively sparse set of entities is extracted from a complex reality.” [13] Sisti described abstraction as “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.” [14] The “lumped models,” as Ziegler referred to them, represent groups of entities based on physical structure. He divided this structural abstraction into four techniques: dropping components, stochastic approximation, coarsening the ranges of descriptive variables, and grouping. This is similar to hierarchical decomposition in traditional object-oriented programming [15]. Fishwick, on the other hand, speaks of process (or behavior) abstraction, and he identified six forms of abstraction: abstraction by representation, induction, total morphism, partial morphism, sensory abstraction, and cerebral abstraction [16].

Another motivation for the use of simulation is that it might be impractical or impossible to experiment with the real-world system. Since purchasing equipment and reconfiguring a factory floor is expensive and time consuming, it is often preferable to conduct simulation experiments beforehand to determine expected throughput, cycle time, etc. When the system does not yet exist it is impossible to experiment with a real system. Examples of this include “modern flexible manufacturing facilities, or strategic nuclear weapons systems.”[10]

Table 1: Some common uses of simulation[12]

1	Simulation enables the study of, and experimentation with, the internal interactions of a complex system, or of a subsystem within a complex system.
2	Informational, organizational and environmental changes can be simulated and the effect of these alterations on the model's behavior can be observed.
3	The knowledge gained in designing a simulation model may be of great value toward suggesting improvement in the system under investigation.
4	By changing simulation inputs and observing the resulting outputs, valuable insight may be obtained into which variables are most important and how variables interact.
5	Simulation can be used as a pedagogical device to reinforce analytic solution methodologies.
6	Simulation can be used to experiment with new designs or policies prior to implementation, so as to prepare for what may happen.
7	Simulation can be used to verify analytic solutions.

The impracticality of experimenting with the real system is generally the case with military simulations. Clearly, it is (at best) impractical to obliterate much of the surface of the Earth in order to explore the effects of nuclear war. Likewise, an enemy army is unlikely to participate in a number of friendly rehearsals of a given plan before the battle. Using historical data is often of little help in determining the effectiveness of a new weapon system or a given strategy. The aspect of military operations that makes historical analysis difficult is that much of the area of operations is damaged or destroyed and many of the observers/participants are killed during a battle. In addition,

the “experiment” can be run only once, while in simulation we can conduct the battle many times, gathering enough data with which to base a statistically significant conclusion.

Hoeber states that models should always “shed light.”[17] He asserted that simulations do this in the following ways:

- The process of constructing the model should increase the understanding of the system by both the model builder and the client,
- Models can aid in making choices, and
- Models sometimes give answers[17].

It is important to note that the absolute number that comes out of a simulation is often of little importance. It is the relative outputs of the various alternatives (experiments) that matter in most cases.

Another pitfall of simulation use is when a single simulation run (experiment) is used to make decisions. 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[10]. 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. Running hundreds of simulation experiments will decrease the size of the confidence interval around output parameters, but Law and Kelton asserted that this will give a false sense of precision.

In any discussion of simulation, the terms entity, attribute, state, event, and activity are used. In simulation, an *entity* is "an object of interest in the system." [12] *Attributes* are properties of entities. *State* of the system is the "minimal collection of information with which [the system's] future state can be uniquely predicted in the absence of chance events."[11] In other words, the state of the system is the information needed to completely describe the system at any point in time. Carson and Banks described an *event* as an "instantaneous occurrence that *may* change the state of the system."[12] (Italics added by author.) Pooch and Wall went further in saying that if the state of the system has not changed, no event has occurred[11]. An *activity* is like an event, but it occurs over some length of time, rather than at an instant in time.

Simulations can be characterized in a number of ways: static vs. dynamic, deterministic vs. stochastic, continuous vs. discrete, open vs. closed, and terminating vs. non-terminating [10, 11]. Each of these methods of characterizing a system or simulation is discussed below.

1. Static vs. Dynamic Simulations

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.

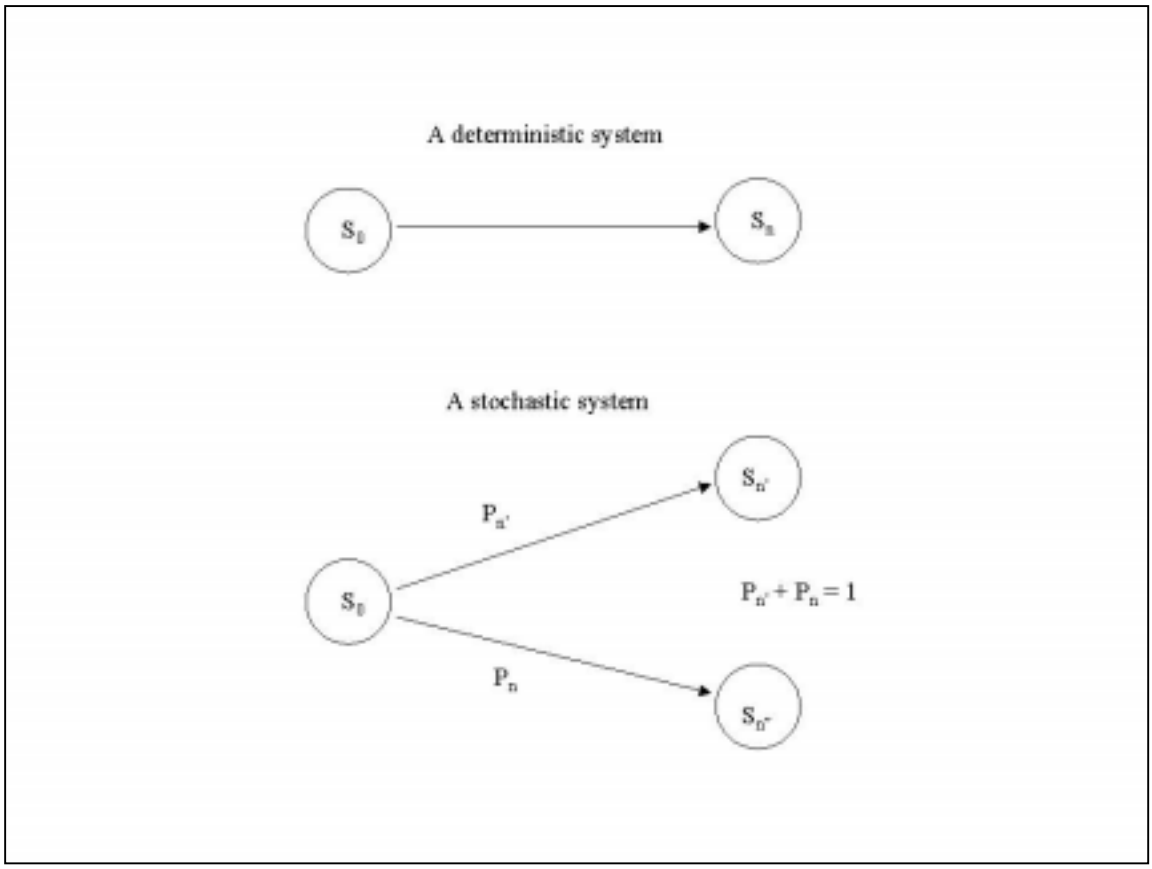


Figure 1: Deterministic vs. stochastic system [11]

2. Deterministic vs. Discrete Simulations

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[18]. In a stochastic system, there is some degree of randomness in the system. Figure 1, shows the difference between a deterministic and stochastic 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.

3. Continuous vs. Discrete Simulations

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.”[10] A discrete simulation is one in which the state variables change instantaneously only at discrete sets of points in time[12]. An example of such a system is Automated Teller Machine, in which a transaction happens instantaneously. Continuous simulations are those in which parameters can be described by a series of differential equations[19].

4. Open vs. Closed Simulations

In a closed system, all state changes are driven by endogenous activities, or those activities that are internal to the system. In an open system, the state of the system changes in response to both endogenous and exogenous activities.

5. Terminating vs. Non-Terminating

A terminating simulation is one for which “there is a ‘natural’ event E that specifies the length of each run (replication).”[10] There are many examples of terminating simulations. The objective of a bank simulation might be to determine how long customers wait to be served during the lunch period. In this case, the simulation would naturally terminate at a time like 1:30 PM. In a military simulation, a natural

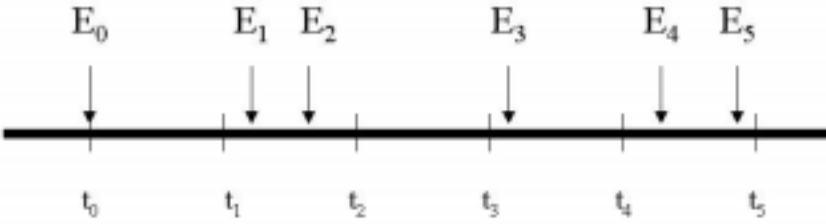
termination condition is when one side wins or loses, where loss might be defined as a certain percentage of the force has been destroyed. In a terminating simulation, "*since the initial conditions… generally affect the desired measures of performance*, these conditions should be representative of those for the actual system."[10]

A non-terminating simulation is one in which there is no natural event E at which time the simulation run should stop. The output of such a simulation is the *steady-state* value of some output parameter. Once again, the initial conditions will affect the value of the desired output variables for some period of time, the *warm-up period*, before the simulation reaches steady state. Steady state is reached when "successive system performance measurements are statistically indistinguishable."[11] In other words, when

$$X_n \sim F_n(\cdot|I_0) \rightarrow F(\cdot)_{n\rightarrow\infty}$$

where X_n is the sampled probability distribution of the output variable which is distributed according to distribution F. Steady state is reached when the sampled distribution X is independent of the initial conditions, I_0. The warm-up period allows all system queues to fill to the level expected in the system when it is running "normally." An example of non-terminating simulations might be simulations of communications (e.g., phone) systems, computer networks, or continuous (three shift) manufacturing facilities.

The distinction between terminating and non-terminating simulation has significant impact on the analysis of the output of the simulation. Analysis of terminating simulations involves running a number of replications of the simulation with a set of initial conditions, and computing standard measures of uncertainty, such as



In an event-driven simulation, events E0 through E5 would be executed at the times indicated along the time line. In a time-driven simulation, events E1 and E2 would both be executed at time t2, event E3 would be executed at time t4, and events E4 and E5 would be executed at time t5. Note that the simulation must wait, doing nothing, between times t2 and t3, even though no events occur during that time step.

Figure 2: Event-driven vs. time-step simulation [18]

means and variances, of the output variables. Analysis of steady-state simulations requires the uncertain, and often *ad hoc*, determination of when steady state is reached. Then information collected before steady state is discarded, and the simulation is run beyond steady state for many iterations. Then the output values are grouped into batches, and statistical analyses are performed on the batches[10-12, 20].

6. Methods of Advancing Time

In a discrete simulation, time is advanced in one of two methods, by time steps and by event steps. Event-step simulation is known at discrete event simulation, and it is discussed in the next section. In time-step simulation, the simulation time is incremented in fixed amounts, as shown in Figure 2. "In the time-step method, the next step is determined exogenous to the simulation. Events are not simulated at their exact time of occurrence, but are aggregated to the end (or beginning) of each interval."[19] While the time-step method has the advantage of simplicity, it can cause

> large approximation errors in large time steps and the possible inefficiencies when using very small time steps [of time steps with no events]. Furthermore, variable [time step] resolution cannot be achieved by merely changing the time-step size without also changing the object representation in the simulation, because time steps that are too small can cause integrality problems, and time-steps that are too large may require too many implicit assumptions about what occurred during the time step.[19]

7. Discrete Event Simulation

Discrete event simulation (DES) "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."[10] In DES, as events are being executed, they may create future events. For instance, when a part in a manufacturing system reaches some workstation (arrival event), the system would estimate when the processing of that part at that workstation will be completed. Then a departure event would be scheduled for that time. Events are added to the event queue asynchronously, and events are ordered

on the queue by their time stamp. Events are removed from the queue in order of their time stamps. When an event is taken from the queue, the simulation clock is advanced to the time of that event. This allows the simulation to move very rapidly through times in which there are no events, to "fast forward" to the next interesting activity.

8. Data-Driven Simulation vs. Input Analysis

When using a simulation the analyst must specify the stream of inputs that will be applied to the simulation. Most often conducting statistical analysis of historical inputs to the real system results in a generator for these input streams. The analyst fits these historical inputs to a collection of probability distributions, and these distributions are used to generate inputs stochastically. This is called *input analysis*. Less often an actual historical stream of inputs is fed to the simulation. This is called *data-driven simulation* [21, 22].

9. On-Line Simulation

Davis has been researching the use of simulations during the operation of manufacturing systems [22-25]. Andersson and Olsson proposed a very similar use of simulation during the operation of a customer order driven assembly line [26]. The use of simulation during military operations was echoed recently by Ferren: "When much-faster-than-real-time simulation becomes practical, the warfighter would have a predictive tool available, online in the field, and integral to their weapons, mobility, and communication systems." [27]

Davis distinguished between on-line planning and off-line planning. In off-line planning, the goal of the analyst is to predict the performance of some system given a set of design parameters (inputs and/or initial conditions). The analyst uses a validated simulation to experiment with different values for the design parameters to determine the best configuration of the system. Then the analyst computes statistical estimates of one or more performance measures[21, 22]. All of the analysis is done before the real system is configured; it is off-line. In most non-trivial applications, there are a large number of design parameters and performance measures. Conducting detailed analyses of multiple values of numerous design parameters can quickly become intractable. There are a number of techniques designed to decrease the volume of the search space [28]; however, "determining the optimum set of values is not a trivial task, and optimality often represents a property that simply cannot be established or verified." [22] Due to the coarse nature of off-line analysis, Davis proposed on-line simulation as a method of improving the performance of real-time systems [23, 24].

Davis has been pioneering on-line simulation as a technique to facilitate flexible manufacturing systems (FMS). As in the command and control domain, which is the focus of this dissertation, an FMS must specify a control policy in addition to state variables and state transitions. This control policy is selected from a number of possible policies, and it can be thought of as a possible course of action. Some of the inputs to a simulation are exogenous, but others are endogenous. These endogenous inputs are generated by the control policy. In open-loop systems, the control policy is specified *a*

priori. In closed loop systems, the endogenous inputs are determined by the control policy *and the current state of the system* [22].

Control policies are analogous to courses of action (COAs) in military operations (described below). Once the COA is chosen (with the help of off-line simulations), the selected COA can be thought of as the control policy for the upcoming operation. Davis asserted that the operating characteristics of FMS change constantly with time. Similarly the shape of a battlefield changes constantly with time.

Davis also discussed a concept that will be critical in the design of the methodology proposed by this dissertation: *autovalidation.* This is a process that compares the result of the on-line simulation over time against the real operation of the FMS and updates to simulation to improve its accuracy [22]. Davis admitted that there are no theoretically sound methods of performing autovalidation and asserted that this is an open research issue. Surdu and Pooch discussed the need for such a feedback mechanism and one proposed approach [29]. This approach will be described fully in Chapter III.

Ideally in a proactive on-line simulation environment, the system should be constantly simulating each of the possible control policies to determine which one the real system should adopt. Clearly these simulation experiments must be generated and conducted much faster than real time in order for such a system to be useful. It takes some amount of time, t, to launch each of the N simulations (where N is the number of control policies to be evaluated). Between the time that experiment n is launched and $n+1$ is launched, the state of the real system has changed. It is desirable that experiment

n+1 consider all the available information. Since *t* can never be zero, all of the *N* experiments may have different initial conditions. This places the comparison of their results on a shaky statistical foundation [22].

A research issue described by Collier [30] is the automated generation of courses of action for the system to consider. The section on classical AI planning (below) will discuss several attempts to automatically generate course of action. Fiebig and Hayes [31] and Fiebig, Hayes, and Schlabach [32] described a system for generating courses of action in the military domain. Their system generated many courses of action using genetic algorithms. In each generation (see the discussion of genetic algorithms below), the courses of action were evaluated by a coarse, low-fidelity simulation. They also used a scheme to ensure that new courses of action generated through crossover and mutation were in fact different than existing courses of action.

There is some controversy over how many iterations of a simulation are necessary in order to reach useful conclusions. Davis asserted that thousands of runs of an on-line simulation are necessary [22]. On the other hand, Law and Kelton asserted that ten to fifteen repetitions give sufficiently reasonable confidence intervals and that increased numbers of iterations gives a false sense of accuracy not justifiable in most simulations [10]. In Davis' prototype on-line simulation systems, he maintained a running average of the last 50, 100, 250, 500, 1000, and 2000 most recent trials. Ideally, the confidence interval of the larger sample sizes should be contained in the confidence intervals of the smaller sample sizes, but this will not be true if the system is in a transient mode (i.e., not in steady state). As the sample size is increased, when the

confidence interval of a larger sample size was not contained in that of a smaller sample size, the smaller sample size was used to estimate the measure of performance. While Davis admitted this was an *ad hoc* technique, he has built a number of prototype systems that demonstrate the success of the approach [23, 24, 33].

C. Military Modeling and Simulation

The military has been using war games to train officers for centuries. *Go* and chess were originally war games; although, their current stylized appearance might belie this. The Prussians were the first "modern" army to incorporate war gaming into their training and planning, and the Germans used war games to plan their successful invasion of France in 1940. The Japanese used war games to plan their attack on Pearl Harbor and their abortive attack on the island of Midway. All these war games used cardboard, plastic, wooden, or metal pieces to represent the various units involved[34].

The Department of Defense (DoD) has developed a large number of computer-based combat simulations. These fall into three basic categories: high-level analysis, low-level analysis, and training. High-level analysis models, like the Theater-Level Campaign Modeler[19] and Institute for Defense Analysis Tactical Warfighter (IDA TACWAR)[17] were used to conduct large-scale analysis of needs, threats, forces, etc. These models provided input on force structure and national military strategy. Low-level analysis models were used to identify the effects of munitions, identify requirements of new weapon systems, and determine vulnerabilities of weapon and vehicle systems.

Training simulations consist of crew simulators (such as flight simulators and tank simulators), known as virtual simulations, and constructive simulations. Constructive simulations are analogous to the paper/wood/metal manual simulations used before computers. These have been used to train commanders and staffs in preparation for war. While these simulations demonstrated face validity[10, 11], none have ever been fully validated. There have been attempts to use some of these training simulations to forecast the outcome of combat operations, but the simulations were not designed or built for that level of fidelity, so their output is of dubious worth. In addition, most users of military models do not understand that it is dangerous to make decisions based on a single replication.

In order to provide the maximum number of staffs and units access to computer simulation models, large simulation centers have been constructed on most bases. The Army pioneered the Distributed Interactive Simulation (DIS) protocol [35] to facilitate the linking of virtual and constructive simulations into a seamless training environment. These simulations, however, are difficult to use during combat operations for a variety of reasons. They require a great deal of contractor support to run the simulation center. For large exercises, contractors are usually hired to create the terrain and force databases used by the simulation – and the various simulations rarely share database formats. Training simulations generally require a large number of special-purpose workstations. For instance, JANUS[5] (a model for company-level operations) runs on Hewlett-Packard, Unix workstations, while most Army units have PCs. Finally, training

exercises using computer simulations often require a large number of participants in addition to the training audience (the target of the training exercise).

Hoeber talked about levels of resolution, and characterized military simulations as one-on-one, few-on-one, few-on-few, many-on-many, and theater. As the level of resolution progresses from one-on-one to theater, the complexity of the model and the interactions between entities increases. An example of a one-on-one/few-on-one simulation is the Unit Conduct of Fire Trainer (UCOFT) used to train crews of M-1 tanks and M-2 or M-3 Bradley Fighting Vehicles. Within Hoeber's many-on-many classification there are various degrees of aggregation from "entity-level" models, in which each entity represents a single vehicle, to large aggregate-level simulations, in which an entity might represent as much as a battalion (approximately 600 soldiers) or brigade (approximately 2400 soldiers)[17].

The reason for higher levels of aggregation is to make it possible to simulate operations over a wider area, longer time, etc. While it is theoretically possible to model a three-month campaign in Europe between NATO and the Warsaw Pact representing each vehicle and soldier as entities, it is obvious that it would take an inordinate amount of time to complete even a single replication. As shown in Figure 3, as the level of abstraction/aggregation increases, the level of detail decreases. The appropriate level of abstraction should be determined by the purpose of the simulation. When forecasting the outcomes of theater-level engagements, it is not important which individual tanks are destroyed. The general maxim in military command and control is that commanders fight with units two levels below them in the hierarchy. In other words, brigade

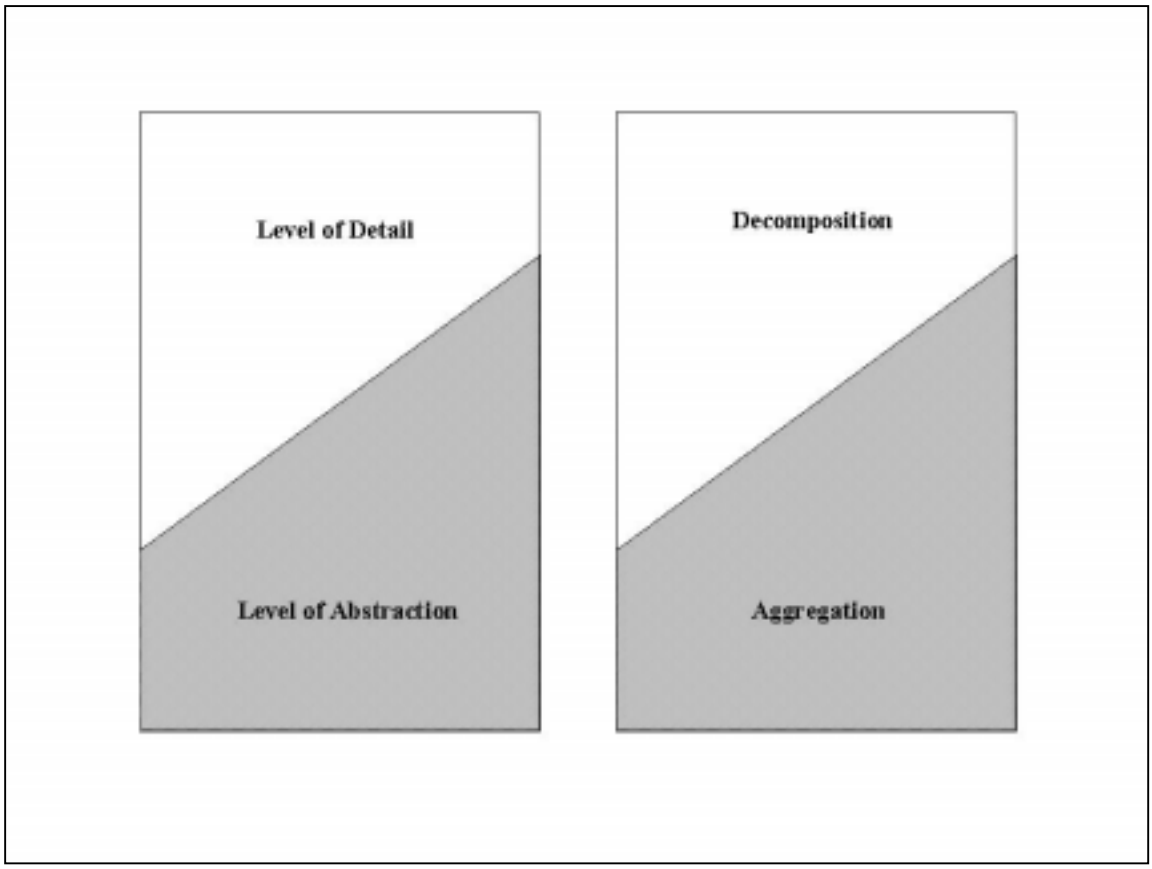


Figure 3: Model relationships [35]

commanders are concerned about companies (the hierarchy being brigades, battalions, companies, platoons) not platoons or tanks.

The military community has developed a large number of simulations for training and analysis, such as the Corps Battle Simulation (CBS), Brigade/Battalion Battle Simulation (BBS), JANUS, and Modular Semi-Automated Forces (ModSAF)[5-7]. As stated earlier, while many of these are excellent products, most are unsuitable for use during an operation for a number of reasons, including large pre-exercise preparation,

specialized hardware, large numbers of required participants, and large numbers of required workstations. Surdu, Haines, and Pooch[8] enumerated the desirable capabilities for an operationally focused simulation to be used during operations, and they include:

- The simulation must be executable from a single workstation by a single user. During ongoing operations, operations centers are crowded, bandwidth is limited, and contractor support is limited. A simulation that cannot be run by a single person on a single workstation would represent a significant burden to an already-busy staff.
- The simulation must be executable on low-cost, open-system, multi-platform environments. While the methodology proposed in this paper concentrates on military applications, an operationally focused simulation is also well suited for emergency management, disaster relief, fighting forest fires, etc. Often the local police and fire units tasked with handling these types of emergencies only have low-end hardware.
- 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 capable of running very fast during planning and rehearsals and running in near real time during operations. (Ferren echoed the need for faster-than-real-time simulation for this application when he asserted that operationally focused simulations should “run at one hundred (or one million) times real-time speed, the sorts of simulations that now run at one hundredth real time.” [27])

- The simulation must be able to receive and answer queries from external agents. This capability allows external software agents to use the operationally focused simulation to help monitor the current, ongoing operation for deviations from the plan.
- If needed, multiple simulations should be capable of operating together. While there is no immediate need for multiple, cooperating simulations, this simulation should be compliant with known, accepted protocols so that this capability is not precluded if it is needed.
- The simulation should be based on an aggregate-level model. In military operations, the basic rule of thumb is that commanders fight with units two levels below them: brigade commanders fight with companies, battalion commanders fight with platoons, etc. This level of abstraction is sufficient for the users of the simulation; therefore, in a desire to be able to run much faster than real time, the simulation need not be entity-level.

Surdu, Haines, and Pooch described a prototype simulation implementation that meets these requirements. In addition, their use of VMAP-2™ terrain databases addresses the issue of exercise setup time and cost. This methodology does not rely on the simulation developed by Surdu, Haines, and Pooch; any simulation that meets these requirements could support this proposed methodology for using simulations during operations.

Blais and Garrabrants echoed many of these desirable characteristics of operationally focused simulations [36]. They described an exercise in which an aggregate-level Marine Corps simulation was used to support mission planning in “a

major staff exercise." Their list of lessons learned from this experiment is similar to that proposed by Surdu, Haines, and Pooch. Blais and Garrabrants asserted that high levels of aggregation were necessary in order to make the simulation usable by a single operator on a single workstation. The requirement that the simulation be capable of running much faster than real time was confirmed. They also discuss the interaction of the planning staff with the simulation run, and their description seems very rehearsal-like. (This will be discussed more fully in the section Military Planning and Problem Solving Process.) Other issues raised by Blais and Garrabrants not discussed by Surdu, Haines, and Pooch were rapid scenario design and flexible, pre-defined measures of effectiveness by which to judge a plan.

In addition, it is essential that an operationally focused simulation is capable of interfacing directly with Army command and control systems. This capability has been kludged into a number of systems and has been demonstrated in several exercises. For instance, the Corps Battlefield Simulation (CBS) was linked to Army Battle Command Systems (ABCS) as part of Army Experiment 4 in 1998 [37] and again for Prairie Warrior 98 [38]. The fact that this integration had to be done more than once is evidence of the *ad hoc* nature of this linkage. The need for simulation as part of ABCS was recognized in the Capstone Requirements Document for ABCS [9]. To be useful in the operational environment, a simulation must be designed to interface with ABCS.

One government-developed simulation that does not currently have all the properties described but which might be appropriately modified to do so is ModSAF[39]. While ModSAF and its proposed follow-on product, OneSAF, are entity-

level simulations, their Distributed Interactive Simulation (DIS)[35] and Persistent Object Protocol (POP) protocols could be wrapped in an "agent" to manage the receipt and answer of subscriptions and queries. ModSAF is not inherently cross-platform, but it has been ported to a variety of Unix and Linux platforms, and the GUI (which communicates with the simulation via UDP/IP messages) might be rewritten in a language like Tcl/Tk or Java to provide this capability.

Colonel Collier, chief of the Army Modeling and Simulation Office (AMSO) described a vision for modeling and simulation in planning and operations by the year 2020 [30]. Collier proposed that military simulations could generate courses of action (COAs) for the commander to consider. Under this concept, based on the current state, a number of simulation experiments would be made, the results would be analyzed, and a small number of candidate COAs would be presented to the commander. Collier proposed this as a method of compressing the planning time line of U.S. Army units and increasing their agility and flexibility. Collier also argued that current, traditional simulations are insufficient for this task, since the user must input COAs and scenario information. He proposed that the simulation be able to read the current state of the operation from digitized command and control systems and automatically generate courses of action to consider. As discussed in Chapter III, the research described in this dissertation is a necessary, but not sufficient, condition for this vision of the year 2020.

Fishwick, Kim, and Lee also asserted that simulations are useful in military COA development and analysis [40]. The prototype system they described used simulations to choose the best route for aircraft trying to avoid enemy anti-aircraft systems. The

prototype was built to account for a considerable degree of uncertainty while selecting one COA over another. An interesting aspect of their approach is the use of an indifference amount.

> When two alternatives are close, we may not care if we erroneously choose one system (the one that may be slightly worse) over the other (the one that is slightly better). Thus, given a correct selection probability P and the indifference amount D, the method calculates how many more replications are necessary to make a selection with the probability of at least P, the expected score of the selected alternative will be no smaller than by a margin of D… A smaller D will produce more accurate results, but with many more replications [40]

Lee and Fishwick used a similar approach to generate and choose COAs for computer generated forces in military simulations [41]. The purpose of the system they described was to make plans for entities in simulation that were not controlled by humans. After the Situation Analyzer generated a number of possible COAs (stored in a tree structure), the COA Tree Simulator simulated each possible COA and chose the COA with the best rating (against some utility function). Lee and Fishwick asserted that the use of simulation is ideal for COA analysis, because "simulation provides a uniform method [for evaluating COAs] without resorting to ad hoc methods." [41]

D. Attrition Models

Attrition models are a means of assessing losses in personnel, weapons, and equipment as a result of combat operations. Attrition is a euphemism for killing people and destroying materiel. The most widely known and understood attrition models are Lanchester Equations and Osipov Equations. Another attrition model is the Dupuy

Qualitative Judgement Model (QJM). There is some dispute about whether Lanchester or Osipov created their attrition models first, whether they based their work on each other's work, or whether they developed their models independently[42]. Lanchester Equations are a series of differential equations that are used to estimate attrition. Lanchester proposed the *square law* (which he asserted fit modern combat) and the *linear law* (for ancient combat)[43]. The square law for modern combat assumed that forces are capable of concentrating their effects at critical points, while the linear law assumed that this is not possible. Bracken showed, however, that the linear law, supposedly meant for ancient combat, fit the Battle of the Bulge (December 1944) better than the square law[44]. Scales [43] also described a third formulation of Lanchester Equations, the *logarithmic law*, which he asserted better predicted casualties "at the extremes" (i.e., many targets and very few firers or very few targets and many firers).

Many proprietary systems such as Calibrated Differential Equation Methodology (CADEM) and Attrition Calculator (ATCAL) are based on Lanchester Equations[19], and there have been many proposed variants to Lanchester Equations [45]. There is an extensive body of literature around Lanchester Equations. Depending on whether the author is attempting to promote a new variant, various papers demonstrate the validity or lack of validity of one or more of the Lanchester "laws."[42-44, 46-49] While Lanchester equations do reasonably well at predicting losses from frontal engagements, they do not predict well circumstances involving flank attacks, surprise, etc.[50] Osipov's equations make some effort to take these circumstances into account.

The Qualitative Judgment Model (QJM) was originally developed to help Dupuy analyze military history (he is a noted military historian). In the two books that describe this method[51, 52], Dupuy developed a series of equations that predicted personnel and equipment losses due to engagements. Dupuy accounted for weather, terrain, surprise, etc. He also attempted to account for intangibles that are most often ignored in military modeling and simulation, such as leadership, training, morale, logistics, momentum, intelligence, relative technological superiority, and initiative. These concepts were lumped into a rating, called Combat Effectiveness Value, or CEV. The problem with this approach is that the assignment of the various coefficients is not well defined. While it is easy to massage these coefficients to reach the correct answer for historical battles, it is unclear how to assign these values to predict the outcomes of future battles.

None of the attrition models proposed is perfect. Each of them has been shown to give correct or incorrect answers on historical data, depending on the circumstances. This makes the choice of an attrition model for a combat simulation difficult.

E. How This Research Applies to Information Warfare

Denning described information warfare in the following way:

> An offensive operation aims to increase the value of a start resource to the offense while decreasing its value to the defense. A defensive operation seeks t counter the potential loss of value. Information warfare is a “win-lose” activity. It is about “warfare” in the most general sense of conflict, encompassing certain types of crime as well as military operations. [53]

This definition, as well as the bulk of the book from which it was extracted, concentrates on the unauthorized use of an enemy’s information (offensive), the destruction of the

enemy's information infrastructure (offensive), or the protection of friendly assets against those activities by the enemy (defensive). Carver described a number of successful and potential attacks on the U.S. military information infrastructure [54], so information warfare is being waged currently. Nitzberg's definition of information warfare is not radically different [55]. The Army's field manual on information operations used the following definition:

> Continuous military operations within the [military information environment] that enable, enhance and protect the friendly force's ability to collect, process and act on information to achieve an advantage across the full range of military operations; [information operations] include interacting with the [global information environment] and exploiting or denying an adversary's information and decision capabilities [56].

This definition makes explicit mention of the *use* of information (e.g., decision capabilities) as a major component of information warfare.

Information warfare papers and books do not appear much different than books and papers on computer security; they concentrate on attacking and defending computer systems. Computer security literature concentrates on the computers (hosts) and networks; information warfare is a broader topic that subsumes computer security. Information warfare also includes such non-computer topics as psychological operations, conventional forms of espionage, and print media exploitation [53].

Information warfare should be more than just computer security. It should also include the more effective or efficient use of information assets. This has been recognized in the vision of the newly established Information Technology and Operations Center at the United States Military Academy at West Point as a "recognized center of excellence in acquiring, using, managing, and protecting information for the

Army." [57] On the digitized battlefield, the U.S. Army now has the capability to overwhelm any decision-maker with more information than that person can process [58]. McLendon stated, "It seems the greater our capability to process information, the more information there is to process." [59] Use of information should be considered a separate part of a three-part information warfare concept rather than lumped in with offensive operations and then given short shrift. The purpose of offensive operations, according to Bunker is information synergy, which "results in a military force gaining battlefield advantage by fusing together the individual contributions of its components into something greater than the sum of its parts… [allowing] for faster OODA (Observe-Orient-Decide-Act) loops, reaction times and decision cycles to take place." [60] It is this synergy, the increased worth and salience of information to the commander at the right time, that the research described in this dissertation is meant to address.

There is increased realization that information warfare will be a deciding factor in dominating maneuver in the 21st Century. McKitrick, et al., asserted that "the military traditionally has viewed information services, including intelligence and communications, as supporting inputs to the actual warfare functions of fire, maneuver, strike, and the like. However, information warfare might not always be a supporting function; it might take a leading role in future campaigns." [61] McKitrick, et al., also predicted the development of "dominant battlefield awareness (DBA)," combined with increased abilities for information processing and dissemination [61], to decrease decision times and dominate future battlefields. Again this addressed the notion that information warfare also involves the increased ability to *use* information. According

the McLendon, discussing decision cycles, “we will use information warfare to expand the adversary’s and compress our own action loops.” [59]

F. Military Planning and Problem Solving Process

Operationally focused simulations are defined in this research to be simulations designed to be used during actual operations. One way the use of an operationally focused simulation will help with situational awareness is by helping to prevent information overload. Bateman described the problem of the various digitized tools feeding the commander and his staff with more information than they can process[58]. Operationally focused simulations, as part of the larger system described in Chapter III, draw the commander's attention to aspects of the current operation that may lead to failure. This helps the commander and staff focus on important information and screen out data that is unimportant to the decision-making process. Ultimately, this will help the commander keep his decision cycle faster than that of the enemy [58].

The conduct of military operations generally consists of planning, rehearsal, execution, and after action review. These are not distinct phases, since most of these actions occur concurrently and continuously. It is helpful, however, to treat each as a distinct and separable phase for purposes of discussion. Simulation technology in the form of operationally focused simulations, can be applied in each of these phases.

1. Planning

During the planning phase, staffs develop courses of action (COAs). The current method, as outlined in U.S. Army Field Manual FM 101-5 and U.S. Army Command and General Staff College ST 100-9, is an *ad hoc* process involving members of the staff discussing the various COAs[62, 63]. Each phase of the operation is analyzed according to an *action-reaction-counteraction* paradigm. This *ad hoc* method has numerous problems, some of which will be addressed in turn.

The effectiveness of the wargaming process is highly dependent on the skill of the commander and the individual staff members. As discussed earlier, it is doubtful that a large percentage of members of a planning staff have the feel of the battlefield that Clausewitz [4] described. There are a large number of time and space relationships that must be considered when going through the *action-reaction-counteraction* drill, and there are no tools to help staff members do this well.

The effectiveness of an *action-reaction-counteraction* analysis of COAs is also dependent to a large extent on the interaction between the various members of the planning staff. The reality of the current personnel management policies is that a staff rarely has time to coalesce. Except for *lock-ins* and *ramp-ups* for deployments to large-scale training exercises, personnel rotations ensure that a fair portion of a planning staff will be new to the group.

Finally, the same officers who develop the COAs are the ones who analyze them for strengths and weaknesses and determine the criteria used to evaluate the COAs. Despite the best intentions, the members of the planning staff carry with them personal

biases as to which plan is better than others. This notion of the developers also being the evaluators can lead to *group think [64]*, in which the decision developed by the group is unduly affected by a desire to conform. Given a bias toward one COA, it is easy to manipulate the criteria, weights on the various criteria, and resultant decision support matrix to support the pre-ordained "best" COA. This bias may be manifested consciously or unconsciously, but it is clearly a risk associated with this *ad hoc* procedure. In the current planning process, once the formal decision briefing to the commander commences, no one in the staff is likely to openly oppose the staff's COA recommendation. Normally, only a forceful commander, assistant commander, or chief of staff can counter this group think.

Operationally focused simulations provide powerful new tools to the planning process. As part of this process, the staff can enter enemy and friendly COAs and then simulate them to assess their effectiveness. The results of numerous simulation experiments can then be used as an evaluation criterion for the staff and commander to evaluate the courses of action. The use of simulations will provide better feedback with higher granularity than current procedures. It will highlight problems, especially synchronization issues, within the proposed COAs. The end result is a timely, more-accurate assessment of the effectiveness of the proposed COAs.

It is true that the use of a simulation is not a panacea. The parameters used to initialize the simulation can be biased. The attrition model can be inaccurate. The staff can still propose "straw man" plans. (The adaptive nature of the system described in Chapter III will help ensure that over time the simulation's parameters will tend toward

"reality.") Given these potential pitfalls, however, an operationally focused simulation would still provide a more accurate, rigorous assessment of COAs than the current, manual process.

The problem of straw-man plans is difficult to solve. Currently a staff usually proposes two valid courses of action and one "throw away," since the commander usually wants three choices; therefore, the staff only considers two viable COAs. This is due to time constraints; there is usually insufficient time to adequately analyze three COAs. A staff armed with a valid simulation with which to conduct COA analysis will be able to adequately analyze more viable COAs -- and do a better job of analyzing the COAs -- than under the current, manual, *ad hoc* method. *While the manual method was appropriate in an industrial-age Army, it is no longer appropriate for an information-age Army that needs to make better decisions faster than the enemy.*

An additional advantage of a simulation-based process is that the commander can conduct experiments in parallel with his planning staff. Previously in this chapter, requirements for an operationally focused simulation were described. One of these was that it be capable of being operated by a single user on a single workstation. The commander can experiment with one or more COAs, conducting mission and COA analysis himself, while his planning staff works on the same ones or others.

If time permits during military operations, the planning staff explores possible alternative actions during the operation (branches) and follow-on operations (sequels). Simulation of the plan makes it much easier for the commander and planning staff to explore more branches and sequels, in more detail, and with greater fidelity. There is

little time to conduct analysis of branches and sequels in the current procedure. As a result, only the most likely, and maybe the most dangerous, branches and sequels can be explored -- and that analysis is often superficial. With the operationally focused simulation, these branches and sequels can be quickly simulated to provide feedback to the planners.

Finally, having the operationally focused simulation at multiple echelons will speed the planning cycle. Once a division headquarters has completed the plan, for instance, they could transmit the plan file electronically to each of the subordinate brigades. The Brigade planning staff can then delete entities that do not pertain to them at the brigade level, partially disaggregate the entities in the division plan that do pertain to them at the brigade level, and begin to flesh out the brigade plan. Once again, this aids in U.S. Army forces making faster decisions than an enemy. If lower level headquarters need to spend less time recopying overlays and redrawing plans created at higher headquarters and more time conducting mission and COA analysis, the planning cycle of U.S. forces can be compressed without degrading the effectiveness of the process.

2. Rehearsal

Once a COA has been chosen, it is developed into a full plan and that plan is rehearsed. The purpose of a rehearsal is to identify synchronization issues and to make sure that everyone fully understands the plan. Certain rehearsals (e.g., fire support (artillery), close air support (helicopters and airplanes), nuclear-biological-chemical (NBC), and mobility/counter-mobility/survivability) are difficult to conduct over sand

tables and maps. Clearly simulation would be an asset for these types of rehearsals; however, a simulation-based rehearsal would also be useful for the traditional, maneuver-centric rehearsal as well. A simulation that can be halted at will could facilitate a rehearsal just as large sand tables and map boards do today.

A significant advantage of a simulation-based rehearsal is that it could potentially be distributed geographically. With a number of distributed graphical interfaces connected to the same simulation, the commander and operations officer could control the execution of the playback of the plan while the subordinate commanders and other staff members watched at remote locations. The rehearsal could be conducted without all the key players getting within grenade-burst radius of each other.

3. Execution

After the plan has been chosen, refined, and rehearsed, and the operation commences, the methodology proposed in Chapter III can be used to monitor the progress of the simulated plan compared to the real operation. Intelligent software agents, referred to as Operations Monitors, will compare the progress of the real plan against the simulation of that plan. When significant deviations from the plan occur, the Operations Monitors launch tools that explore the impact of these deviations. Finally the commander is advised if the Operations Monitors determine that the success of the plan is in jeopardy. The remaining chapters of this dissertation focus on the use of simulation during the execution of an operation.

4. After Action Review

After action reviews are important – even during a war. The course of the real operation could be recorded and archived for later review. As time permits, the operation can be "played back" for the key leaders. This would give the commanders and staffs the opportunity to identify synchronization problems or other errors that lead to the final outcome of the operation. During training exercises there are often observer/controllers to dispassionately observe the conduct of planning and operations and provide feedback afterwards. This capability is unlikely to be available during real operations. The use of an operationally focused simulation could help fill this void.

G. Software Agents

There are many different definitions of software agents. In addition, a number of adjectives have been applied to agents, such as autonomous, adaptive, mobile, and intelligent. Franklin and Graesser elected to describe properties of agents rather than try to form one, general definition [65]. The characteristics of agents are shown in Table 2. According to Franklin and Graesser, all programs must satisfy the first four properties in order to be considered software agents. Under this system, agents are classified by their properties. For example, one might construct a mobile, learning agent [65].

Russell and Norvig broke agents into four broad categories: reflexive, reflexive with internal state memory, goal-directed, and utility-based [66]. Reflexive agents make immediate, predefined decisions based on the current state of their environment. A serious drawback of reflexive agents is the potentially large search space to find the proper "reflex" to execute for agents in a complex environment. Many interesting robots have been built based on this idea. The most ardent proponent of reflexive agents is Brooks. Brooks' subsumption architecture is based on the notion that intelligence is built from the lowest-levels up [67-69]. While Brooks asserted that only physical

Table 2: Characteristics of software agents [65]

PROPERTY	MEANING
Reactive	Responds in a timely fashion to changes in the environment
Autonomous	Exercises control over its own actions
Goal-Oriented	Does not simply act in response to the environment
Temporally Continuous	Is a continuously running process
Communicative	Communicates with other agents, perhaps including people
Learning	Changes its behavior based on its previous experience
Mobile	Able to transport itself from one machine to another
Flexible	Actions are not scripted
Character	Believable "personality" and emotional state.

entities, like robots, will eventually develop intelligence, Etzioni responded that software agents, which he called softbots, also have the potential for intelligent behavior [70]. Coen used a combination of Brooks' subsumption theory and classical AI work in constructing the Intelligent Room at MIT's AI Laboratory [71, 72]. This project, using a collection of agents, collectively known as Scatterbrain, was designed to create "rooms that listen to you and watch what you do; rooms you can speak with, gesture to, and interact with in other complex ways." [72] Becket and Badler also used a hybrid of subsumption ideas and traditional high-level control based on symbolic reasoning for path planning [73]. Maes proposed another hybrid approach which interleaved planning with execution [74]. This discussion, of course, begs the question of how to define intelligence, which is not the subject of this research or dissertation. Suffice to say reflexive agents can accomplish useful work.

There are many problem domains in which a good answer cannot be reached just with information about the current state. Russell and Norvig's second class of agent is a reflexive agent with memory. Even these agents have limitations. For instance, if a taxi were at an intersection, no amount of memory or information about the current state would help the taxi decide whether to go left, straight, or right. This requires a goal. Goal directed agents are Russell and Norvig's third classification of agent. In the real world, there are many situations in which people (and agents) have multiple, competing goals. In this case, simple goal-direct behavior may not be sufficient. This leads to the fourth classification of agent, utility-based agents. A utility-based agent attempts to

maximize its utility function by choosing the alternative with the highest utility value [66].

Franklin and Graesser and Russell and Norvig provide just two definitions of agency. Maes[75-77], Coen[71], Nwana[78], and others have different definitions. For purposes of this research the characterization approach of Franklin and Graesser, rather than any one stringent definition, will be used.

1. Rational Agency

In defining and describing software agents, Russell and Norvig defined their fourth, and most "intelligent," type of agent as utility-based [66]. There is a body of literature that discusses rational agents. Essentially rational agents are utility-based or decision-theoretic agents. As Williams, Decker, and Sycara stated "We construe 'decision-theoretic plan' quite broadly to mean a plan intended to be 'good' with respect to some explicit measure of quality (i.e. a value or utility function)" [79]. Rational agents seek to maximize a utility function [66, 80]. Sandholm and Lesser observed that the value of an agent's actions may change over time. An agent may deliberate for a long period of time to determine the best answer if time permits, or the agent may give a quick, crude answer if time is limited. In order to do this, the agent must be able to conduct meta-reasoning about itself. Some approaches to this problem use *ad hoc* meta-level rules. Sandholm and Lesser proposed a method grounded in utility theory. This use of utility functions or probability theory to conduct meta reasoning is known as *decision-theoretic control* [81]. Sandholm and Lesser used utility functions to perform

rational decision making while Doan and Haddawy used probability theory for their decision-theoretic planning [82].

2. Reasoning Under Uncertainty

In most practical problems, the people (or agents) tasked to make decisions do not have perfect knowledge about the state of the world or problem domain. In this case, they are forced to make decisions that take into account this uncertainty. A number of useful approaches to reasoning under uncertainty have been proposed. Among these are *ad hoc* methods, such as EMYCIN [83] certainty factors and Dempster-Shafer Theory, Bayesian systems, Fuzzy Logic, Artificial Neural Networks, and Genetic Algorithms. Each one of these techniques will be discussed briefly in this section.

MYCIN, an expert system for the diagnosis of blood diseases, was one of the first and most famous success stories for expert systems. One of the things that made MYCIN so successful was that the expert system shell was separated from the logic of diagnosis. This empty version of MYCIN, empty MYCIN or EMYCIN, could then be used for building other expert systems. The other major contribution of MYCIN was the use of certainty factors to account for reasoning under uncertainty [83]. The calculus used to combine these certainty factors was *ad hoc*, not based on any theory. In addition, the combining of certainties in an EMYCIN system was not commutative; "given the same evidence it is possible to get different [certainty factors] depending on the order in which the rules are invoked." [84] In addition, Henkind and Harrison pointed out that the measures of belief and disbelief in an EMYCIN system "tend to

converge quickly to the asymptote 1, while [certainty factors] stay near 0. Therefore, the calculi do not perform well if there is a great deal of evidence to be combined." [84]

At first glance Dempster-Shafer seems to be much like EMYCIN. The Dempster-Schafer Theory, like EMYCIN, is intuitively pleasing, and it allows evidence to be relevant to subsets of theories rather than just to single beliefs. Dempster-Shafer assumes that all evidence is independent, an assumption that is often not true in real applications. Like EMYCIN, there is no theoretical foundation for the rule used to combine evidence. [84]

Bayesian Systems, also known as probabilistic reasoning systems, are based on the notion of conditional probabilities [66]. To use this method, a set of prior probabilities, or *priors*, must be gathered. Prior probabilities can be determined by means of statistical analysis if sufficient information about a population is known [84]. Probabilities based on intuition, known as subjective probabilities may also be used, but of course the accuracy of the intuition impacts on the reliability of the system. Bayes' Rule is used to compute the likelihood of some event based on evidence and the conditional probabilities determined by prior probabilities. Bayes' Rule states that:

$$P(v \mid a_1, a_2, \ldots a_n) = P(a_1, a_2, \ldots a_n \mid v)\, P(v) / P(a_1, a_2, \ldots a_n)$$

In other words, the probability of some event, *v*, given that events a_1 through a_n have occurred is the probability of a_1 through a_n given that event *v* has occurred times the probability of even v divided by the probability of events a_1 through a_n without regard to event *v* (known as prior probabilities). Computing all of these prior probabilities is often

intractable. While this method has strong foundations in probability theory, it can lead to large computational complexity in real applications. One approach to solving this problem is the use of Naïve Bayes, which uses the conditional independence assumption. This assumption states that the probability of a conjunction of events a_1, a_2, … a_n is just the product of the individual probabilities [85]:

$$P(a_1, a_2, \ldots a_n \mid v) = \Pi_i P(a_i \mid v)$$

This simplifying assumption greatly decreases the number of prior probabilities and the computational complexity of using Bayesian networks to solve real problems.

Fuzzy logic is based on the notion that objects belong in sets. Unlike traditional set theory, however, the boundaries between sets are gradual, or fuzzy. The height of people provides a good example of fuzzy set membership. Clearly a seven-foot tall man would fit in the TALL set, and a four-foot tall man would fit in the SHORT set. In what set would a man be if he was five feet, six inches? Five feet, ten inches? Some people might regard him as short, while others would consider him tall. The boundary between TALL and SHORT is not crisp. An arbitrary, crisp boundary can be chosen, say five feet, ten inches. The problem with this crisp approach is that in many real-world applications, as the value of a parameter changes a very small amount but passes some crisp threshold, the behavior of the system can be radically different. Fuzzy set theory uses the notion of degree of membership in a set (a real number between 0 and 1, inclusive) to deal with imprecision or uncertainty. The degree of membership in the TALL set for a man who is five feet, six inches tall might be 0.5 [84]. The real benefit of fuzzy logic is that it provides a way to “develop cost-effective approximate solutions to complex

problems by exploiting the tolerance of imprecision." [86] Fuzzy logic also provides a good way to help push through the knowledge acquisition bottleneck [83] associated with building expert systems. When building a fuzzy rule base, the knowledge engineer can allow the human expert to describe the problem in semantic terms, like "very vast," "somewhat fast," and "not fast." Ragsdale, et al., demonstrated the power of fuzzy rule bases by building a prototype system that determined the best location for reconnaissance assets in an area of interest [87].

Artificial Neural Networks (ANNs) are modeled on the network of cells in the human brain. Many nodes are connected together. When the ANN is stimulated with some input, information is propagated through the network and an answer – often a classification – is output. The weights on the various connections determine how the information moves through the network. ANNs are often classified as a machine learning technique, because they undergo a training process in which the weights are adjusted. This learning can either be *supervised* or *unsupervised* [88]. During training, ANNs use a gradient descent algorithm to move toward a (hopefully) global minimum. Like other gradient descent search techniques, ANNs are susceptible to falling into local minima. A number of techniques, such as momentum, stochastic gradient descent, and the use of multiple networks, have been used to minimize this danger[85]. During training, credit for correct classifications and blame for incorrect classifications must be attributed to the many connections between nodes in the network. The most common technique for updating connection weights is the *back-propagation algorithm*. ANNs

are also susceptible to “overfitting the training examples at the cost of decreasing generalization accuracy over other unseen examples.” [85]

Genetic Algorithms (GAs), like ANNs, are inspired by biological systems. GAs operate on a pool of hypotheses (or plans or classifications). During each training iteration, or epoch, each member of the pool is evaluated by means of some fitness function. Some of the most fit are carried forward into the hypothesis pool for the next iteration, called the next generation. Other members of the next generation are created through operators called *mutation* and *crossover*. In mutation, small random changes to a hypothesis are used to create a hypothesis in the next generation. In crossover, two hypotheses are split apart and recombined to create two new members of the next generation. Of vital importance to GAs is the selection of a representation of the problem that facilitates the application of a fitness function, mutation operators, and crossover operators [85]. Usually information is represented as bit strings, but other higher-level representations have been demonstrated. Porto, Fogel, and Fogel showed how combinations of plan fragments could represent plans for computer-generated forces in a combat model [89]. The computer generated forces could use GAs to learn the best course of action by conducting mutation and crossover of the plans, treating plan fragments as atomic entities [89].

3. Agent Collaboration

Ygge and Akkermans made the case that multi-agent systems may be more effective than single, monolithic systems or conventional systems (those that do not employ any AI or Machine Learning techniques) [90]. The use of multiple agents to

solve a single problem often involves some form of collaboration between the software agents. Several collaboration schemes have been proposed and demonstrated. This section will outline a number of those schemes.

Genesereth and Ketchpel asserted "communication language standards facilitate the creation of interoperable software by decoupling implementation from interface." [91] They recommended the Agent Communication Language (ACL), developed as part of the ARPA Knowledge Sharing Effort. ACL includes three parts: vocabulary, an inner language, and an outer language. The inner language, Knowledge Interchange Format (KIF), is an enhanced version of first order predicate logic. It is these KIF expressions on which the agent operates. KIF expressions are wrapped in Knowledge Query and Manipulation Language (KQML) expressions. The first term in KQML expressions describes the type of communication represented by the expression (e.g., tell, perform, reply, ask-if, subscribe, etc.) [91]. Genesereth and Ketchpel were interested in large numbers of autonomous software agents who had need of sharing large amounts of data. In many smaller, well-defined domains, the use of less formal, less verbose, more *ad hoc* techniques may be useful.

Bradshaw seemed to reject the KIF/KQML architecture as not robust enough for practical use. Instead, he has developed a Common Object Request Broker Architecture (CORBA) approach, called KAoS [92]. Bradshaw's architecture took advantage of many CORBA facilities to allow agents to conduct conversations, advertise services, and look for agents that provide needed services. *Proxy agents* represented remote agents in the local network. *Mediation agents* were used to communicate with non-KaoS agents.

Bradshaw asserted that the KAoS architecture made it easy to implement agents that conducted useful work.

In a system involving many collaborating agents, it is likely that there would be a number of different communication languages and protocols used. To address this problem, Finin, Fritzson, and McKay proposed the use of *communication facilitators* [93]. Among other things, the purpose of these facilitators was to translate from one protocol to another. The global, common language in this approach was KQML. One of the fields in a KQML message included the language in which the content of the message was encoded. A facilitator used this information and knowledge of what language the agent used to translate incoming and outgoing messages.

Decker and Lesser proposed a technique, which they called Generalized Partial Global Planning, in which the various, distributed planning agents communicated coordination relationships with other agents [94]. In addition, the distributed agents communicated their goals to the others. As each of the distributed planning agents made its local plans it could take into account these relationships and goals.

Ekenberg, Boman, and Danielson used agents organized into a two-level hierarchy [95]. Each of the lower-level agents provided imprecise, possibly conflicting information to the higher-level, decision-making agent. It was the job of this decision-making agent to resolve the conflicts in the information it was provided. The decision-making agent used the information from the lower-level agents to determine a small set of admissible strategies or choices. Ekenberg, Boman, and Danielson used a technique they called proportion, which gradually decreased the “confidence intervals” around

information to determine when strategies became inadmissible. Their use of a senior/subordinate hierarchy is interesting for two reasons: it greatly reduced the collaboration overhead on the lower-level agents, and it closely modeled the structure of human organizations. While there is no mention of this by Ekenberg, Boman, and Danielson, there is nothing in their architecture to preclude several decision-making agents being lower-level agents to another, higher decision-making agent.

The Retsina system developed by Sycara, et al., involved a dynamic agent organization that was tailored to the specific task [96]. When an *interface agent* communicated a task to a *task agent*, the task agent decomposed the task into subtasks. The task agent then invoked another task agent for each subtask. This continued until the task was broken into atomic pieces. In Retsina there were a finite number of task agents. Task agents used matchmaker agents to determine the location of other task agents to assign subtasks. This allowed agents to enter and depart the system at will, making Retsina dynamic and robust. The various task agents interleaved planning, scheduling, coordination, and execution. These task agents communicated with *information agents*. The information agents did the work of interfacing with databases and the Internet in response to queries from task agents. All communication was done through KQML messages. As the lowest level task agents accomplished their missions, the results were passed back to the task agent that invoked them. Eventually the final result was returned to the interface agent.

The system described by Ekenberg, Boman, and Danielson (with a static two-level hierarchy) and that described by Sycara, et al., (with a dynamic hierarchy for each

task) involved organizing the agents for some purpose. While it seems that many agent researchers are in favor of completely autonomous agents collaborating with each other in some collegial way, “in multi-agent domains, agents can never be completely autonomous from one another: necessary sacrifices in autonomy must be made in order to allow interaction.” [97] Anderson and Evans asserted that in many domains complete autonomy is not desirable. The reason that humans have formed organizations and hierarchies is to streamline and facilitate communication and coordination. As Anderson and Evans stated, in many domains “more can be accomplished in unison than could be done autonomously.” [97] Anderson and Evans advocated the concept of flexible autonomy in which agents sometimes work autonomously and other times collaborate. Lesser echoed these ideas by claiming that “large ensembles of semi-autonomous, intelligent agents working together is [sic] emerging as an important model for building the next generation of sophisticated software applications.” [98] To facilitate the collaboration between agents, Lesser proposed Generalize Partial Global Planning, as discussed earlier. The organization of agents into collections or hierarchies does not invalidate the agent paradigm.

H. Applications of AI to Mission Planning and Control

The focus of this section is on software agents (and other AI techniques) in the planning process. This is a broad discussion of how AI techniques can be used to facilitate military planning; however, it will begin with a discussion of the basic concepts of classical AI planning.

1. Brief Discussion of Classical AI Planning

Since the beginnings of AI, researchers have developed algorithms for automated plan generation. Yang cites several examples of successful planning algorithms and packages in domains from office building construction to spacecraft activity planning [99].

A problem with many planning algorithms is that "difficult problems cause goal interactions. The operators used to solve one sub-problem may interfere with the solution to a previous sub-problem. Most problems require an intertwined plan in which multiple sub-problems are worked on simultaneously." [100] This is known as *nonlinear planning*. Another motivation for nonlinear planning is that it avoids early commitment to the ordering of subtasks until the last possible instant. This helps avoid replanning and unwanted side effects of actions [101]. Three well-known nonlinear planners are Nonlin, Tweak, and POP [48, 100]. POP, as described by Russell and Norvig, is an off-line planner (the planning is done before execution begins) involving the incremental satisfaction of preconditions for achieving the desired goal [66]. Sub-goals that have been achieved are "protected," so that the satisfaction of another sub-goal does not undo the first sub-goal.

Beyond these basic algorithms, Yang divides extensions of planning algorithms into two basic categories: problem decomposition and hierarchical decomposition [48]. Problem decomposition involves breaking down a goal into sub-goals that can be solved separately (and concurrently). This is what Russell and Norvig describe as the divide and conquer approach: "We solve each sub-problem separately, and then combine it into

the rest of the solution." [66] Problem decomposition still requires the kinds of conflict resolution necessary in POP, but it also provides some opportunity for optimization of tasks that have the same sub-tasks [48]. As an example, if the goal was "enjoy a honeymoon and raise a baby," possible solutions to the two sub-problems might be "get married and go on a honeymoon" and "get married and have a baby." Clearly one would not have to get married twice to accomplish the overall goal, so these two plans can be merged, eliminating redundancy [66].

Hierarchical decomposition involves solving for a goal by hierarchically decreasing the level of abstraction of a solution. The goal is initially solved with very coarse, abstract operations. Once the main issues of satisfying the goal have been solved, these abstract operators are decomposed into sub-plans, invoking necessary conflict resolution. Each of these less-abstract operations is broken down successively until a concrete, executable plan is developed [66, 100].

Other interesting planning techniques include triangle tables, metaplanning, macro-operators, and case-based planning [100]. Triangle tables are used to repair a plan that will fail due to unexpected events during execution. Metaplanning allows the planner to reason about the problem and also about its own planning process. Macro-operators are combinations of commonly used sequences of primitive operators. Case-based planning involves the use of previously built plans as a starting point for solving new and similar problems.

Brooks [67-69], Agre and Chapman [102], and Kaelbling and Rosenschein [100, 103] have argued that there is no need for planning, and that the agent should use the

observable situation to determine the correct reaction. It is doubtful that all intelligent decisions can be made merely from knowledge of the current situation [66], and a number of hybrid systems that capitalize on reactive and deliberate planning have been developed [71, 72].

There have been a number of recent advances and refinements in planning technology. Boutilier, Dean, and Hanks proposed a planning method involving Markov Decision Processes to account for planning under uncertainty [104]. Drabble and Tate propose treating planners as situated software agents in which planning is just part of their job of task assignment, planning, execution, and control. In the O-Plan architecture proposed by Drabble and Tate, the situated planning agents operated in a hierarchy of strategic (for analysis and direction), tactical (for planning and scheduling), and operational (for enactment and control) agents [105]. Fink and Veloso and Veloso et al., built a learning system, PRODIGY, which "includes a planning algorithm and procedures for learning control knowledge to improve planning efficiency and plan quality [106, 107]. Stone and Veloso extended PRODIGY to allow interleaving of planning and execution [108]. This was needed if the user wanted to begin execution while the system was still planning or if some actions needed to be taken in order to complete the planning. Finally, Fishwick, Kim, and Lee proposed a planning system used for course of action generation and selection for entities within a constructive simulation [40].

All of the planning approaches discussed, except the reactive methods, are state-based; the world is represented as a series of states, and the planer reasons with "state

predicates." Lansky proposed an approach, called action-based planning in which the planner reasons with action-based constraints [109]. While she admited that the state-based and action-based approaches have equivalent inference capabilities, she asserted that the action-based representation made inference easier in coordination and logistical problem domains.

2. Data Fusion

One of the most obvious and pressing applications of AI techniques in Mission Planning and Control is the difficult issue of data fusion. Data fusion is an ongoing research area with its own series of conferences and journals. As discussed previously, digitized sensors and information systems have rapidly created a problem of potential information overload. In addition, they have created a problem of data fusion. As an example, a number of redundant sensors (e.g., a soldier with binoculars, satellite imagery, moving target indicator (MTI) equipped aircraft, and helicopters with advanced sensor packages) with differing accuracies and areas of coverage might each report a formation of enemy vehicles. Air Force aircraft and satellites might have reported generic vehicles with some velocity vectors in radians; the helicopter might have reported 25 tanks moving in some direction in degrees; and the scout might have reported 22 tanks and 14 armored personnel carriers. Each report's time might be different by up to ten minutes, for example, and its location might be different by up to 1000 meters. Were these reports of four different formations or of one formation at different times and locations? Another example of data fusion is the aggregation of various reports of, for instance, battalion-sized formations. Given these battalion-sized

formations, which battalions belong to the same brigades (a higher-echelon unit), and where are the support units likely to be? Both problems are difficult, and today human intelligence analysts solve them.

Ntuen, et al. stated the data fusion problem well when they wrote

> Current systems on the digital battlefield emit information in a heterogeneous format, with different units, sources, and with channing frequency in space and time. The commander needs tools to integrate information based on context, form and content. Informaiton used for decision making needs to be presented as a homogenous set, in scalable format, with a composite (single) performance metric. [110]

The role of data fusion is to "combine related data from multiple sources to provide enhanced information." [111] Data fusion has many military applications, including automated target recognition for smart weapons, guidance systems for autonomous or semiautonomous vehicles, battlefield surveillance, and battlefield identification. Thomopoulos, Hilands, and Chen demonstrated a data fusion system which fused information from forward looking infrared (FLIR) and laser radar (LADAR) systems for extracting targets from a noisy (Guassian noise and LogNormal noise) scene [112]. This system performed better than either the FLIR or LADAR separately. Non-military applications of data fusion include robotics, monitoring and control of automated manufacturing processes, and medical applications. [113]

Many interesting data fusion prototypes have been proposed. Rosenblatt and Thorpe demonstrated their Distributed Architecture for Mobile Navigation (DAMN), an approach to data fusion for controlling autonomous vehicles [114]. Their system was based on the subsumption (reactive) architecture proposed by Brooks [67-69] (and described earlier). Murphy demonstrated Sensor Fusion Effects (SFX) for fusing sensor

data on teleoperated robots [115]. Mayk, et al., proposed a distributed system for ensuring that multiple platforms on the battlefield have the same relevant common picture of the battlefield [116]. Their system involved distributed data base technology as well as a rule-based expert system to resolve database consistency problems.

Promising machine learning techniques could be applied to this problem, such as decision trees, artificial neural networks, Bayesian learning, instance-based learning, genetic algorithms, rule learning, analytical learning, and various hybrid approaches [85]. Three approaches that seem particularly useful in this domain are fuzzy rule bases, Knowledge Based Artificial Neural Networks (KBANN) and Explanation Based Learning.

Fuzzy rule bases were discussed earlier. Gibson, Hall, and Stover demonstrated a fuzzy logic architecture for "control and sensing resources, fusion of data for tracking, automatic object recognition, control of system resources and elements, and automated situation assessment." [117] It is this last application, automated situation assessment, which is of most interest here. Gibson, Hall, and Stover showed how their methodology fit in the DoD Joint Directors of Laboratories (JDL) Data Fusion Process Model. This model specifies four levels of processing:

- Object refinement – combining location, parametric, and identity information to achieve refined representations of individual objects,
- Situation refinement – developing a description of relationships among objects and events,

- Threat refinement – projecting the current situation into the future to draw inferences about "enemy threats, friendly and enemy vulnerabilities, and opportunities for operations," and
- Process refinement – meta-processing about the system [113].

Gibson, Hall, and Stover asserted that fuzzy logic could be applied to all four levels of processing [117]. Oxenham, Kewley, and Nelson also developed a fuzzy rule-based expert system as a decision support tool for humans that fits in the JDL model. Their system used fuzzy generalized *modus ponens* inference mechanism; however, they departed from the theory of fuzzy logic by using some aspects of Dempster-Shafer and Possibility Theory. As a result, "due to the manner in which [they] aggregate the necessity [the extent to which the evidence supports the claim] and possibility [the extent to which the evidence does not contradict the claim] during the rule processing and in resolving conflict, [their] overall inference mechanism does not coincide with" either of the three theories [111].

KBANN [85], or Expert Network [118, 119], is based on the notion that a domain theory (probably rules articulated by domain experts like intelligence analysts) could be used to initialize a learning system and that examples could be used to tune the domain theory. The learning algorithm used in KBANN is an artificial neural network (ANN). KBANN specifies an algorithm for converting Horn clauses into an ANN. Then examples are presented to the ANN to refine the domain theory[85, 120, 121]. Kuncicky, Hruska, and Lacher showed that the inference capabilities of expert systems and ANNs are equivalent [118]; however, this approach "typically generalizes more

accurately than Backpropagation when given an approximately correct domain theory." [85] Mitchell asserted, "Although it is sometime possible to map from the learned network weights back to a refined set of Horn clauses, in the general case this is problematic because some weight settings have no direct horn clause analog." [85] Still, the possibility that there might be a way to explain to a human expert how the ANN is making decisions is compelling. It might prove useful to show the refined domain theory to human experts, let them refine the theory further, and reinitialize the ANN with this refinement. Over several iterations and sufficient training examples, an excellent classifier would be created. Wann and Thomopoulos demonstrated multi-sensor parallel data fusion architecture using several ANNs; however, there has been no attempt to seed such a system with an appropriate domain theory.

Like KBANN, in Explanation-Based Learning (EBL) the learning system begins with a domain theory represented as a series of Horn clauses. The system is then exposed to a number of training examples, which are used to modify the domain theory. EBL can be thought of as "theory-guided generalization of examples," in that it uses the domain theory to make a rational generalization of the examples. EBL distinguishes between relevant and irrelevant attributes of the training examples, helping to reduce the computational complexity of the final, learned theory. EBL can also be thought of as "example-guided reformulation of theories," in that it modifies the domain theory so that it can be used more efficiently later. It does this by combining clauses of the domain theory so that it can classify new instances in a single step. EBL has been criticized as just restating what the learner already knows, the initial domain theory. While this may

be true, the initial domain theory might be inefficient to use [85]. Given a relatively accurate domain theory by an intelligence analyst and many training examples, EBL might well result in a good classifier for data fusion.

3. Situation Analysis and Decision Support

Maes coined the term Personal Digital Assistants (PDAs) [76]. PDAs are Maes' vision of software agents working alongside humans rather than as adaptive, "intelligent" interfaces to software [122]. The user trains PDAs through four methods: "looking over the shoulder" of the human, getting direct or indirect user feedback, using examples, and asking the advice of other PDAs. The goal of Maes' approach was to build software agents that act as the user's assistant. Maes demonstrated how such a system could help deal with information overload and also provide decision support [76]. Mitchell, et al., described a personal assistant they built to manage people's calendars [123]. This system used a machine learning technique (decision trees) to learn rules. Over time, the system provided useful assistance to its users. The goal of such PDAs should be to respond to "uncertain and incomplete information, provide reminders to previous (context) scenarios, generate multiple hypotheses and multiple consequences, and… assess decision risks." [110] Hersh also asserted that PDAs should take into account the sustainability of the action they recommend; the utility of a course of action should consider the long-term effects of the decision [124].

If a number of systems used the same scheme to represent the user's profile, as the user was promoted or moved, the user's profile could move as well. While a person promoted to run a larger organization would probably have different information

requirements, the new PDA would not have to begin the process of modeling the user from scratch.

Wong, Wang, and Goh described a fuzzy neural network used to support decisions about managing stock portfolios [125]. This approach was unique in that it used both a fuzzy rule base and a back propagation-trained neural network. The inputs to the fuzzy rule base were fuzzy membership functions (either manually or automatically generated from data), fuzzy rules with weights indicating the credibility of the rules, historical data, and current data. The outputs of the fuzzy rule base became the inputs to the neural network, and the neural network rated the investment potential of a country or particular stock.

Sauter [126] and Hersh [124] described how some decision-makers are intuitive and others are analytical. Sauter asserted that many decision support systems draw on extensive data warehouses. These potentially provide more information than a decision-maker can process. One technique she suggested would aid in presenting information to a user in a manner that would facilitate analysis and decision making was data mining [126]. Data mining often involves a variety of AI or Machine Learning techniques [85].

In recent years the field of adaptive and intelligent user interfaces has increased. Clearly intelligent interfaces can make use of many promising AI techniques. A large number of information gathering agents have been developed to aid in decision support. Zilbertson and Lesser described a system that used a model of the user's decision process [127]. This decision model, called an influence diagram, was implemented as a directed graph, in which the decision to be made had all outgoing edges and the utility

function had all incoming edges. The decision model was used by the system to plan information-gathering activities. When sufficient information was gathered, the system computed the utility of the various options and reported those values to the user.

There are at least two other notable information gathering agent implementations. Rus, Gray, and Kotz described an adaptive agent for gathering information in a heterogeneous networked environment. Their agents were written in an extension of the Tcl scripting language, which has been ported to most architectures [128]. This allowed the agents to move to where information resided, collect the information, and return to the user, reducing bandwidth usually needed for queries across the network. The system described by Decker, et al., involved three types of agents working together to collect information for the user [129]. *Interface agents* interacted with the user. The interface agent informed a *task agent* what mission it must perform. The task agent determined what information it needed to solve the problem and sent KQML queries to the *information agent*. The information agent selected plans from its plan library, scheduled the execution of those plans, and monitored the results. Eventually the requested information was sent to the requesting task agent. The novelty of this approach was that the task agents could be involved in more than just gathering information; they were performing some action on behalf of the user.

There are at least two interesting prototype applications of planning in a military domain. Fiebig and Hayes [31] and Fiebig, Hayes, and Schlabach [32] discussed the use of genetic algorithms in *planning support systems*. These planning support systems are designed to help users formulate plans. “Like an automated planner, it may (but does

not necessarily) perform part of the problem solving for the user. In contrast to totally automated planners, a planning support system always leaves the final control of decisions and solutions in the hands of the user." [31] The system that Fiebig, Hayes, and Schlabach proposed involved the use of genetic algorithms to generate courses of action. A coarse, low-fidelity simulation was then used to select the most promising courses of action to recommend to the user. Hayes, et al., and Liang and Hayes demonstrated a prototype system, CoRAVEN, which uses Bayesian Belief Networks to analyze course of action[130, 131]. Lehner, Vane, and Laskey proposed a knowledge-based approach to generating courses of action in domains (such as military planning) in which there are "far too many legal moves to examine exhaustively" [132]. This method borrowed heavily from game theory.

4. Control of Simulation Entities

There have been a number of attempts to make the entities in simulations more intelligent. The goal of such work has generally been to decrease or eliminate the need for large numbers of operators to control enemy forces in war games. If the goal of few or no enemy controllers is realized, simulations could much more easily be used for decision support, as described previously. This section discusses a number of research efforts in the area of intelligent simulation entities.

One of the first, and most difficult, aspects of intelligent simulation entities is automated plan generation and selection. The work Lee and Fishwick described as part of the Institute for Simulation and Training (IST) Intelligent Autonomous Behavior by Semi-Autonomous Forces in Distributed Interactive Simulation (DIS) project is one

attempt at automated plan generation [41]. Their research, described earlier, focused on plan generation for company-sized forces.

Porto, Fogel, and Fogel demonstrated the use of genetic algorithms for plan generation and selection [89]. In the research they described, they represented plans as sequences of atomic plan segments. Using genetic algorithms, the ModSAF [39] entities generated and selected plans based on tunable fitness functions. The production release of ModSAF allows the control of entities through *behaviors*, which are implemented as crisp rule bases. Porto, Fogel, and Fogel attempted to make this process more dynamic and intelligent through the use of genetic algorithms. A limitation to this work at this point is that they have only implemented this with one entity on each side. It is clear that a large number of entities, operating within a military hierarchy, must be supported in order to make such a system feasible.

When the small airplane crashed into the Rose Garden of the White House in 1994, the subsequent Congressional investigation suggested that the Secret Service needed to take more advantage of simulation technology to aid in contingency planning. As a result, the Secret Service created a provisional organization, consisting of four agents, to explore the usefulness of simulation [133]. (This division was subsequently made a permanent organization within the Secret Service.) In conjunction with Army Research Laboratory and Lawrence Livermore National Laboratories, the Secret Service experimented with the Joint Tactical Simulation (JTS) [134]. The Secret Service found that simulation increased the effectiveness of their planning and training. In separate research Baxter and Hepplewhite proposed an architecture for implementing entities in

simulation as a hierarchy of intelligent agents, and they demonstrated that the behavior of the agent-based entities was superior to that of traditional entities implemented as finite state machines [135]. The purpose of the simulations conducted by the Secret Service and Baxter and Hepplewhite was to conduct training; however, more intelligent simulation entities would also facilitate the use of simulation for mission planning and rehearsal.

Surdu, et al., also showed how entities in simulation could be implemented as intelligent agents and how those agents could be organized into a hierarchy [136]. Using fuzzy rule bases, lower-level agents performed one of the following actions: no action, informed their immediate superior of an action they were taking, or asked permission to take some action. Higher-level agents could overrule subordinates, allow subordinates' judgements to stand, or issue orders. This prototype used platoons as the lowest-level entities, and implemented this methodology for platoons and companies.

Applegate, Elsaesser, and Sanborn also used this hierarchical organization of entities in simulation in their approach to adversarial planning, called Battle Planner [137]. Battle Planner depicted orders to subordinate units as leaves in a plan tree. Not only were the entities organized into a hierarchy, but the planning mechanism was an extension of the hierarchical planners discussed in the earlier section on classical AI planning; the Strategic Planner built abstract, high-level plans, and the Action Managers built detailed, lower-level plans. The Strategic Planner maintained a representation of its current plan as well as an estimate of the enemy's plan. Both levels of planner used knowledge of the current situation to signal when replanning should occur. Plan nodes

had preconditions associated with them, and when these preconditions were violated, the Strategic Planner and/or Action Managers began replanning. Each Action Manager controlled an entity in the simulation, making the entity semi-autonomous. To facilitate this decision-making process, an Action Manager selected from a number of pre-defined "plays" the play its entity would execute. Since the "battle plan's later steps are necessarily contingent on he success of earlier steps and on intelligence gathered during execution… the plan generally will not be complete in detail at any point in the battle." [137]

Ragsdale, et al., also demonstrated an implementation of simulation entities as intelligent agents [87]. In their prototype intelligent agents, using fuzzy rule bases, chose the best positions from which to conduct reconnaissance of enemy positions. The prototype also selected the best routes to and from those positions.

Another approach to modeling the decision processes of agents in simulation was proposed by Luo and Su [138]. Their prototype system, Warfare System, operated at a very high level of abstraction as their goal was to simulate the human decision making process. Warfare System used a fuzzy utility function to choose one of several possible course of action. Luo and Su did not define the source of these possible courses of action. At the level of abstraction described by Luo and Su, it is unclear how their approach would scale to many decision-making entities within the simulation.

Parsons described a system requirement developed by the Center of Naval Analyses (CNA) [139]. This project was based on the aggregate-level maneuver warfare simulation MWARS. MWARS used Lanchester equations [42-47, 49, 50] to resolve

combat situations. The purpose of this research was to simulate human decision making within the simulation model to evaluate various campaign plan alternatives. The design of this system was again based on fuzzy rule bases to assess the alternatives, but it is unclear how alternatives were to be generated in this approach. There is no documentation on whether this system design was ever implemented, so one must conclude that a prototype was never developed.

Becket and Badler proposed a hybrid planning approach between symbolic reasoning and reactive reasoning for the control of intelligent entities in simulation [73]. Their prototype system was designed to conduct path planning in a world filled with obstacles for a single entity (a walking human). In Becket's and Badler's approach, they created a new level of abstraction, the Object Specific Reasoners (OSRs), which communicated between the symbolic reasoner and the reactive behavior modules. The OSRs translated intentions, created by the symbolic reasoner, to actions for the behavior modules to execute.

There have been a number of attempts to make simulation entities more autonomous and intelligent. When this technology matures, this will facilitate military planning and operations. As simulation entities become more autonomous, a single operator will be able to control large numbers of entities. This will decrease the personnel and workstation overhead associated with using simulations to support course of action development and analysis [36]. These semi-autonomous entities can be used to ease the burden on the planning staff of inputting the plan and running the simulation, and it can be used to eliminate the need for enemy controllers.

CHAPTER III

DESIGN

A. Proposed Methodology

The proposed methodology for linking simulations and operations is summarized in Figure 4. The methodology involves the interactions of a number of packages and tools, including an operationally focused simulation (discussed in the previous chapter) and intelligent agents. Each of the various components of the methodology is discussed below.

1. OpSim

The operationally focused simulation runs in near real time, tracking the predicted progress of the plan. The progress of this simulation can be monitored from the web-based GUI. The Operations Monitors (OMs), discussed below, register interest in various entities and events with the simulation, and they query the simulation directly for information.

2. Operations Monitors (OMs)

The OMs are the heart of this methodology. They perform two important functions. They take information from WorldView and update the state of entities in

OpSim, seamlessly re-synchronizing the simulation to the real world. More importantly, they monitor the progress of the simulation and compare it to that of the real operation. When they discover significant deviations between the real world (WorldView) and the simulated world (OpSim), events referred to as Potential Points of Departure (PPDs), they launch one of a number of tools to explore the ramifications of these deviations.

It is important to note that OMs do not take actions with regard to the plan; rather, they explore the ramifications of differences between the real operation and the

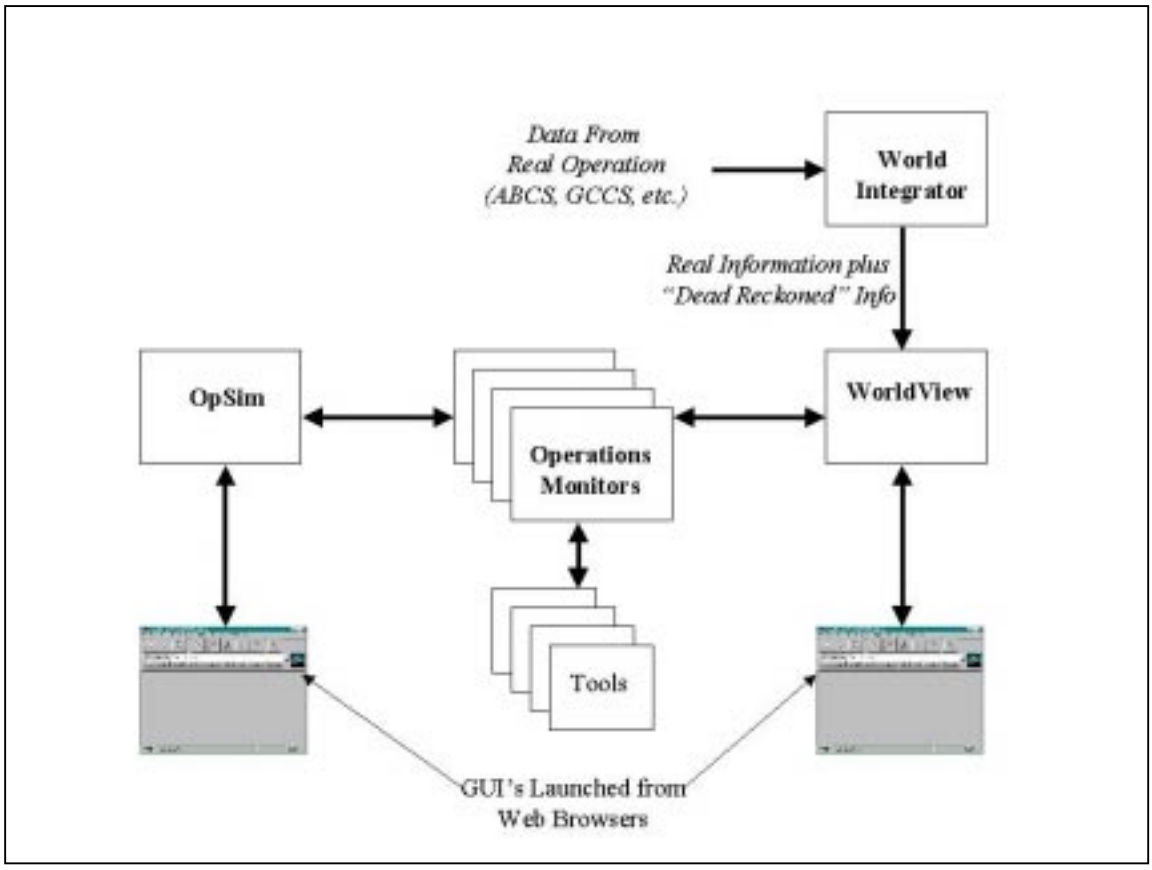


Figure 4: Proposed methodology

planned operation. The job of OMs is to help human decision-makers manage information [74, 76, 77]. OMs should be considered part of the team, not a replacement for decision-makers [140]. OMs make some judgment about the seriousness of any differences and issue advisories to the decision-makers.

Also, it is important to note that OMs must be proactive. It is not sufficient, for example, for an OM to inform decision-makers that some planning timetable has been broken. The OMs must look ahead and inform decision-makers in advance if some goal is unlikely to be met. For instance, if some future event requires that three of five preconditions be met, the OM must determine whether these preconditions are likely and assess the probability that the eventual goal can be accomplished. When this probability becomes "low enough" the OM must inform the decision-maker.

It is important to remember that this methodology is not designed to create a course of action. This methodology is designed to assist humans in planning, but it does not do any planning itself. It is not the job of this system to generate courses of action but to assess their results through simulation (and some artificial intelligence tools) and to advise the commander and staff when the probability of success is low. Because this system does not do the actual planning, many of the problems associated with AI planning systems are not a factor (e.g., they work well in some domains and situations; but not in others; early commitment to the ordering of subtasks that requires re-planning; elimination of redundant steps; and conflict resolution arising from competing goals or sub-goals).

OMs are implemented as rational, utility-based agents. There are a number of useful definitions of "agent." Franklin and Graesser [65] proposed a list of characteristics that describe a piece of software as an agent: autonomous, reactive, goal-oriented, persistent, communicative, adaptive, mobile, and flexible. According to Franklin and Graesser, software which has the first four characteristics is an agent, and software which exhibits some of the other attributes in the list fall into more specialized categories of agents. This method is used in defining OMs within this methodology as agents.

Each OM is interested in only a narrow domain. By focusing each OM on a very narrow domain, the problem of building intelligence into these agents becomes more tractable. The prototype OMs (described in Chapter IV) use simple fuzzy rule bases and simple crisp expert systems to reason about their domain and the current operational situation.

PPDs are not always held in fixed, global knowledge bases. Instead they are domain specific. Each OM has a knowledge base that it uses to analyze the discrepancies between the real operation and the simulated plan. Because of the inherent uncertainty in the knowledge associated with the domains, non-crisp forms of reasoning (or soft computing) are often required. For some OMs, the PPDs may reside in a fixed knowledge base, which may be in the form of rules [83], while the PPDs for others may be in the form of some refinable domain theory [85, 123], or any other representation scheme appropriate for the domain. It is not the intent of this methodology to impose any particular inference mechanism on the OMs. The prototype OMs constructed to

validate this methodology used fuzzy rule bases [86]. With a fuzzy rule base, it is relatively easy to capture the knowledge of domain experts and later explain how the OM made each decision. *The point is that the specific inference mechanism used to implement a particular OM is independent of the overall methodology proposed in this research.*

3. WorldView

The WorldView module is a representation of the real operation. In order to make the job of the OMs easier, the representation of the state of the real operation and the simulated plan should be as similar as possible. WorldView receives information about the state of the real operation through a series of APIs. It then transforms this information into a form that the OMs can easily interpret.

4. WorldIntegrator

WorldIntegrator has the onerous task of monitoring the real operation, processing that information, and passing it to WorldView. In some systems, such as the Global Command and Control System (GCCS), this may involve querying a database. 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 WorldIntegrator to "dead reckon" these intermittent reports and pass them into WorldView. Clearly, when an entity has been "dead reckoned," this must be reflected in the information that WorldView gives to the OMs, perhaps by some measure of confidence.

The tasks of WorldIntegrator and WorldView involve sensor, data, and information fusion. WorldIntegrator must determine when an entity has been unconfirmed long enough that its actions must be dead reckoned. When some sensor reports a similar unit, WorldIntegrator 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 areas. Since no such system is currently available, for purposes of this research a combat model simulates the functions of WorldView and WorldIntegrator in the prototype (see Chapter IV).

5. Tools

This research does not attempt to enumerate all possible, useful tools. Instead, it gives examples of tools and how the OMs might use them. For example, the Enemy OM might note, based on information from WorldView, that there are two enemy mechanized battalions in the area of operations rather than the one assumed during COA analysis. The OM can call a combat attrition model to determine the difference in expected losses, or the OM might merely apply a closed form solution to Lanchester equations to get a quick estimate of expected losses. If it appears that this difference will adversely affect the plan, the Enemy OM will notify the decision-maker.

Similarly, if the Mission and Time OMs note that some ground unit had missed an important phase line by forty-five minutes, they might launch another simulation to explore how this would effect other units. If the effect is minimal, the OM might recommend to the commander that the overall time line be shifted forty-five minutes to resynchronize the simulation.

There is a great deal of interaction between the OMs and OpSim and between the OMs and WorldView. This is conducted through a message-passing protocol. There are two kinds of requests: individual queries and registration of interest (subscriptions). An OM, for example, might send a query to WorldView and OpSim about the status of a particular unit. This is done as an individual query. An OM might also register interest in certain information. For instance, the Troops OM might register for periodic updates of the strengths of units. Registration of interest is preferred, since in an ongoing basis it requires roughly half the number of messages as individual queries. OpSim launches a separate thread to handle each of these registrations.

Note that normally OMs do not make tactical decisions. The purpose of an OM is to explore differences and report findings. The autonomy of the OM lies in its ability to decide when and if to launch other tools. As noted in the DARPA CPoF concept, battlefield visualization tools should be decision-centered. Among other things, this means that these visualization systems "show decision-relevant details, highlight relevant changes, anomalies, [and] exceptions, and portray uncertainties" [3]. These are exactly the pieces of information that this proposed methodology is designed to provide. Visualization is not a tool to show the battlefield in a unique way; visualization is a process that occurs within the heads of the commander and his staff [56]. This proposed methodology provides additional support for this process.

B. Synchronizing the Real Operation with the Simulated Operation

In order for the OMs to adequately compare the real operation with the simulated operation, the two representations must be "close." An axiom in military planning is that no plan survives the first rifle shot. After the operation commences, the plan will certainly diverge to some extent from the real operation. The job of the collection of OMs is to identify when this divergence has become so great that the success of the operation is in jeopardy. The OMs report this concern to the decision-maker (probably with some rating of certainty attached to this conclusion).

Once the decision-maker has been notified that the currently running simulation no longer accurately reflects the state of the actual operation, the simulation should be updated. If the simulated plan is allowed to continue to diverge from the real operation over time they may become almost completely unrelated. Any analysis the OMs would perform at that point would be meaningless. This updating also allows OpSim to better predict where the operation will be at some future time. The problem, however, is to define a synchronization mechanism which is feasible and adaptive. The proposed approach is best illustrated through two examples.

The combat effectiveness of entities (units) in the simulation is characterized by some probability distribution(s). These probability distributions (which might be different for different classes of entities) are used in the analysis of the various COAs during the planning phase, and they are also used to simulate the interactions between the various entities as the simulation is paralleling the operation in near real time.

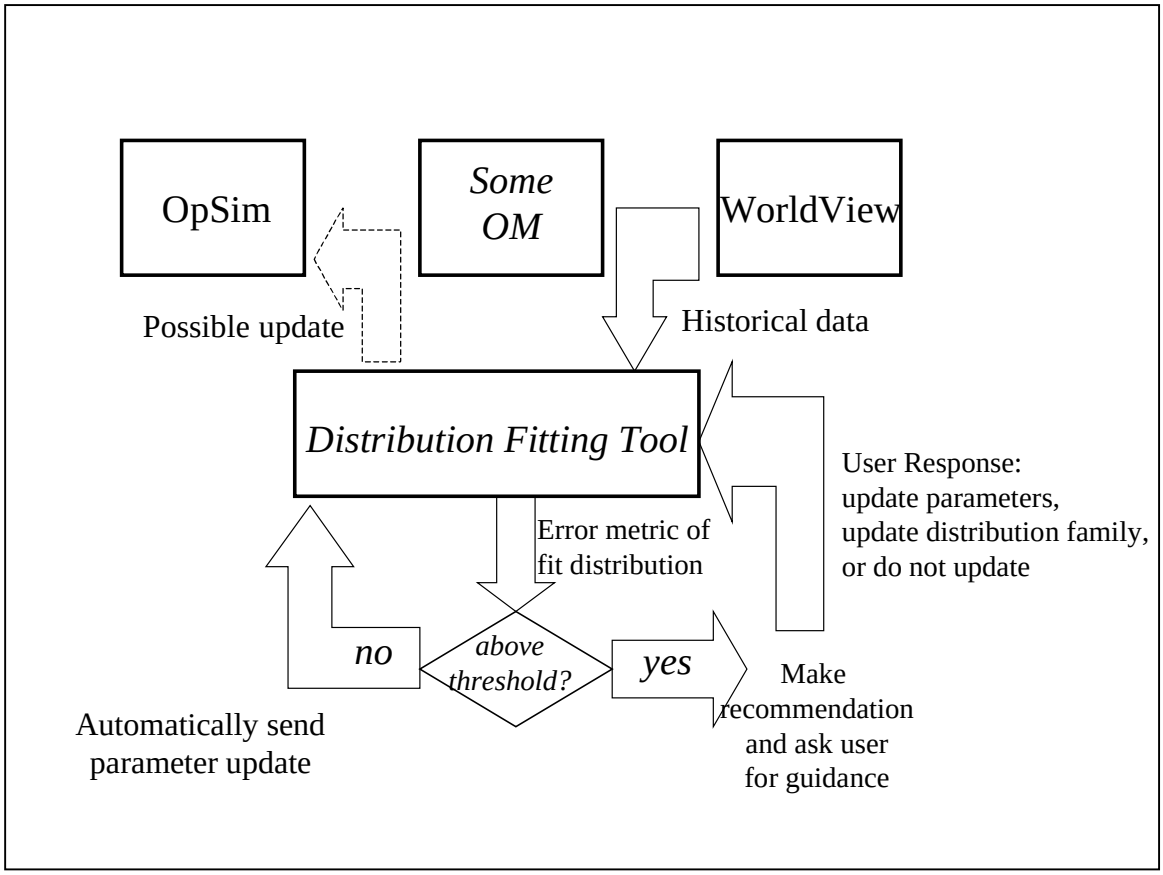


Figure 5: General depiction of synchronization/updating scheme

At the time the OMs determine that the real operation and the planned operation are significantly different, they have a body of historical data on the actual effectiveness of the classes of entities. The OMs must do two things: update the current state (e.g., the number of casualties) and update the future performance of the entities within the simulation. An OM tries to refit the historical data to the family of probability distribution described for that class of entity. There are a number of commercially available distribution-fitting packages that accomplish such a task. They usually use

maximum likelihood estimators (MLEs) to recommend some distribution for a set of data based on which distribution has the smallest p-value or sum of squared errors. When the OM tries to refit the historical data gained thus far in the real operation to the family of distribution defined for the combat effectiveness of the entity, some p-value or sum of square errors will be generated.

If the measure of error is below some threshold, the OM will conclude that the distribution family is correct, but that the parameters (e.g., μ and σ for a Normal distribution) are incorrect. In this case, the OM would automatically send an update message to OpSim. If the error metric is above that threshold, the OM will conclude that the family of distribution chosen is incorrect. At this point, the OM will open a dialog with the user of the system describing the problem. It is then up to the user to decide whether to merely update the parameters of the current distribution with the best possible values, change the family of distribution used, or determine that the results are anomalous and decide that no update is necessary. As this should make the future performance of the simulation better over time, this updating scheme makes the system adaptive. This process is summarized in Figure 5.

As an example, in OpSim combat effectiveness is represented as a distribution family (e.g., Normal or Exponential) with some parameters. Periodically (at a time interval determined by the user, but approximately every five minutes by default) the Results OM queries the simulation and the real world for the historical data on the various classes of entities (e.g., Bradley-equipped mechanized infantry platoons). Using the real world data as the “true” distribution, the Results OM conducts a Kolmogorov-

Smirnov goodness of fit test of the data from the simulation [11, 141]. (The more common Chi-Squared goodness of fit test has also been implemented, but the Kolmogorov-Smirnov test has greater power, in the statistical use of the term [141].) When the data from the simulation fit the real data with an alpha value of 0.95, the Results OM takes no further actions. When the data from the simulation do not fit the real data, the Maximum Likelihood Estimator (MLE) of a variety of distributions is computed for the simulation data. Then sample data streams are generated with each of the computed MLEs, and a Kolmogorov-Smirnov goodness of fit test is computed against the real data. The new distribution that has the best fit is nominated as the correct distribution. If the new distribution is in the same distribution family as the true data, the Results OM tells OpSim to update the parameters of the distribution. In order for the system to remain stable the new value is set to the average of the old value and the suggested value. This allows the parameters to move slowly toward the "true" value, rather than changing rapidly and giving undue importance to transient spikes in the data. In experiments in which the probability distribution is intentionally set to a different family of distribution, the Results OM tells the user what the old family and parameters are as well as the new recommendation, and asks the user to confirm or cancel the update. Over the course of an experiment, the parameters that describe entities in the simulation gradually converge on those of the real operation. Most of the changes occur immediately after small engagements, after which new data is available to the Results OM.

The preceding illustration shows adaptive updating of OpSim for combat effectiveness; however, the methodology is valid for other characteristics as well. Movement rates of classes of entities are also defined in terms of probability distributions (e.g., an average movement rate and some variance). If all the entities (units) of a certain type are moving more slowly than predicted, we would want to do two things: put the entities in their current locations in the real operation and adjust the parameters on the distribution which describes their movement rate. Again, it would be up to the decision-maker to determine if this difference was anomalous (e.g., unseasonable weather, entities beginning movement late, etc.) or required an update to the simulation.

Another, less-technically interesting update of OpSim would occur when the number of entities was significantly different. If, for instance, the plan assumed that the enemy would have three tank battalions, but WorldView indicates the enemy actually has four, OpSim would need to include this additional unit in the future. Adding this unit automatically would be difficult, since an intelligence officer would have to provide OpSim with an estimated plan for this new entity. Initially, the OM might provide OpSim with a plan that extrapolates the entity's velocity vector, but adding new entities would probably need some human intervention.

Prototype Friendly and Enemy Forces OMs have been implemented (see Chapter IV). These are identical, except one looks at the friendly forces and one looks at enemy forces. Each of the Forces OMs periodically compares the number and strength of units (friendly or enemy as the case may be) in the plan (simulation) versus those in the real

operation. Almost as soon as the operation begins, the strengths of the units in the simulation and real operations begin to diverge. In test scenarios additional enemy units have been added to the real world to force the Enemy Forces OM to analyze the impact. The Forces OMs use a fuzzy rule base to determine when the strength and/or number of units are significantly different. When the strength and/or number of units is significantly different, the Forces OM must determine whether this difference is important. It does this by feeding the current, real situation into a simulation (another instantiation of OpSim) and running that simulation to completion some number of times (determined by the user). The mean results of the new simulation are compared with those of the planned operation. Again this comparison is done using a fuzzy rule base. If the Forces OM determines that the impact of the change in strength/number of the units involved significantly impacts on the probability of mission accomplishment (or the end strengths of the friendly forces after the operation), the user is notified. If the difference is insignificant and only involves the strengths of units (e.g., the unit is at 50% strength rather than 75% strength), the Forces Monitor sends an update message to OpSim. If the difference is significant or involves adding or deleting units, a human must make the corrections manually.

The basic idea, therefore, is for one or more OMs to analyze the performance of the simulation with respect to the real operation. The OMs can make small updates in the parameters of OpSim automatically. For larger deviations they query the user for help. The amount of data that needs to be gathered before the differences are significant is unclear. One problem with analysis of military operations is that the experiments are

not reproducible, much of the area of operations is destroyed in the process, and many of the witnesses are killed. One approach to dealing with this issue is for the threshold (used to determine whether to update automatically) to be adaptive. The OMs can use the performance of the simulation after an update to help it adjust the threshold. At some point it might also be appropriate for the correctness of the human user's decision to be used to adjust this threshold as well.

C. How OMs Are Created Dynamically

As stated earlier, OMs focus on a narrow domain. This makes their design and implementation more tractable. OMs exist in a tree-like hierarchy, as suggested by Ekenberg, Boman, and Danielson [95]. When the system is first launched, a manager OM creates the first layer of OMs in the hierarchy. The overall manager is responsible for synthesizing the reports of the agents below it in the hierarchy. Each OM in the hierarchy compares the current situation with the plan, looking at the operation from a particular, narrow perspective. One such taxonomy for OMs in this first layer is the use of the Combat Functions (as defined in FM 100-5): maneuver; fire support; air defense; command and control; intelligence; mobility, counter-mobility, and survivability; and combat service support (logistics and personnel) [142].

These OMs in the first level of the hierarchy have a number of tools (and additional agents) available to them to perform their analysis. Each OM uses some inference mechanism to determine whether to launch additional tools to help with analysis and decide what actions to take. There are basically three types of rules: those

that dispatch other agents or tools, those that report information to other agents, and those that take other actions (e.g., advising the human of problems or updating the simulation as discussed earlier). Some of these rules are domain dependent; they are related to the particular focus of the OM. Other rules are domain independent; they are related to general issues, such as the resources needed by a particular candidate tool.

Domain independent rules take into account RAM usage, time complexity, bandwidth, etc. (as shown in Table 3). These domain independent rules are important, because they help insure that the system does not grind to a halt during times of peak activity during the operations. For instance, the eventual system might have two different path planning algorithms that could be used to determine the impact of rerouting logistics. If one algorithm has a smaller time complexity but is somewhat less accurate than the other algorithm, the OM might choose to use this algorithm if the host on which it is running is already very busy. As mentioned earlier, OMs are utility-based agents, and these domain independent rules provide information used to compute the utility of a particular action.

One possible taxonomy for agents in a second layer of OMs might be along the lines of the Army's METTTC mnemonic (Mission, Enemy, Time, Troops, Terrain and Weather, Civilian Impact). Under this taxonomy, one OM would be looking for differences in the size, strength, and/or composition of the enemy. Another might be looking at effects of terrain and weather.

Table 3: Sample structure of tool classification used by an OM

Domain 1:		
	Tool 1	Memory(N), O(N), Bandwidth(N)
	Tool 2	Memory(N), O(N), Bandwidth(N)
	Tool 3	Memory(N), O(N), Bandwidth(N)
Domain 2:		
	Tool 4	Memory(N), O(N), Bandwidth(N)
	Tool 5	Memory(N), O(N), Bandwidth(N)

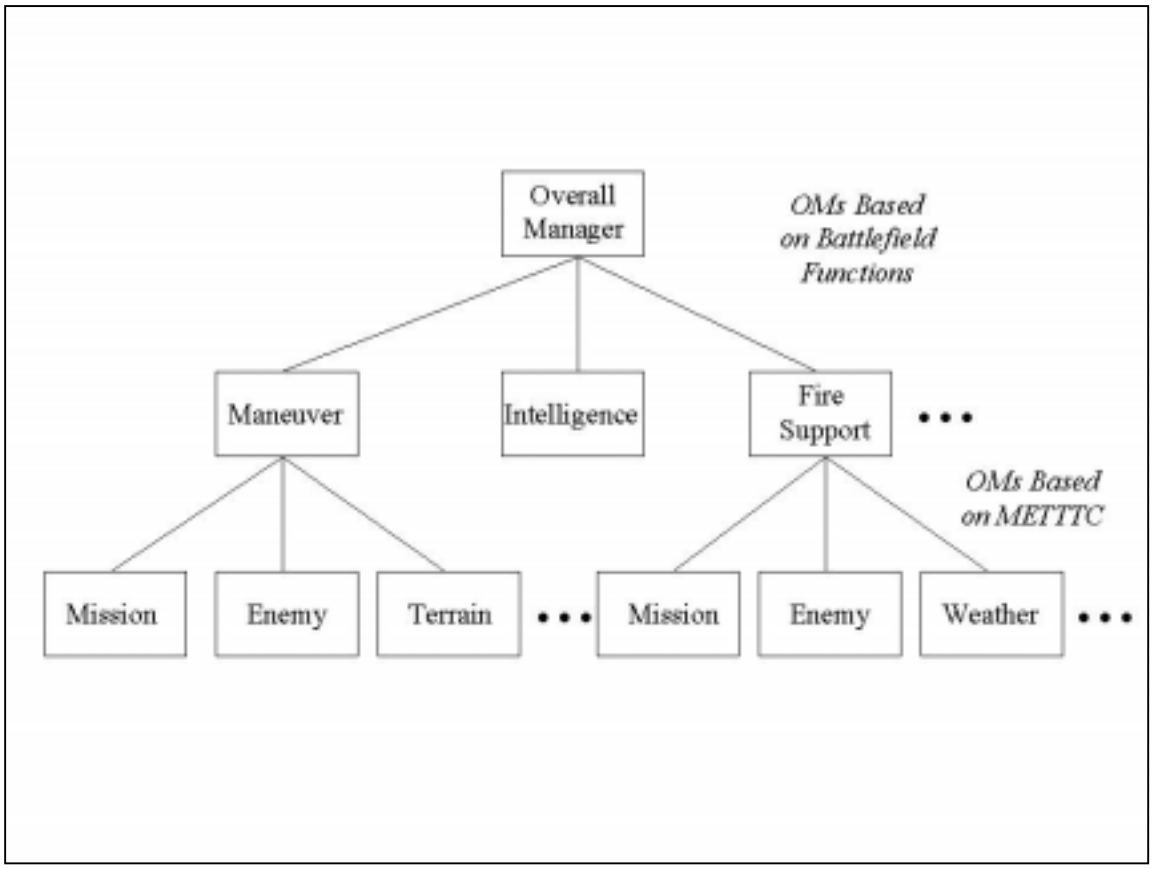


Figure 6: A possible, partially expanded hierarchy of OMs

One possible, partial expansion of an OM hierarchy is shown in Figure 6. Note that an instantiation of a particular class of OM can exist at multiple points in the hierarchy. The only restriction is that an OM cannot call a tool above it in the hierarchy to avoid circular dependencies. Since any OM can create any number of subordinate OMs to aid in its analysis based on the current situation and delete OMs that are no longer necessary, the hierarchy is dynamic.

This discussion of OMs has repeatedly referred to the use of rules. This is for ease of discussion. This methodology neither requires nor relies on any particular reasoning mechanism. As discussed earlier OMs could use a variety of reasoning technologies, including crisp rule-bases, fuzzy rule-bases, machine learning techniques, artificial neural networks, etc. An OM's domain and mission will probably dictate the appropriate reasoning mechanism. OMs must be capable of launching other tools as necessary, making inferences about the current situation, and taking actions as appropriate. This methodology does not specify the technology used to perform these tasks.

To avoid the haphazard and *ad hoc* creation of OMs, a general principle was devised. This is based on something that every commander readily has available: his mission. Army manuals clearly define the exact meaning of various tactical tasks, such as seize, secure, defend in depth, and screen [142]. For instance, while it is not stated in this bullet format, a *seize* mission involves:

- Gain control of a piece of terrain and
- Deploy to prevent its destruction or loss to enemy action.

and *defend in depth* involves:

- Prevent enemy forces from penetrating the rear boundary of the sector,
- Retain flank security, and
- Maintain integrity of effort with parent unit's scheme of maneuver.

While there is not a one-to-one mapping between mission statements and what OMs to create automatically, the mission provides some guidance for the top-level OM. For instance, in the prototype implementation the seize mission was used in an experiment. Some analysis of a generic seize mission by domain experts (field grade Army officers at Texas A&M University) indicated that the Maneuver OM, C2 (Command and Control) OM, and Intelligence OM would be needed for all seize missions. The OMs representing the other Battlefield Functions (Air Defense; Logistics; Fire Support; and Mobility, Counter-mobility, and Survivability) would be needed on a case-by-case basis. A similar analysis was done of defense in sector, and Fire Support OM was added for that mission. Below the top-level OM, the other OMs create sub-OMs as their reasoning mechanism dictates.

CHAPTER IV

IMPLEMENTATION

A. Introduction

In order to demonstrate the efficacy of this methodology it was necessary to implement a prototype based upon the design described in Chapter III. This chapter describes the implemented prototype. The basic components of the proposed methodology are OpSim, a representation of the actual operation (*WorldView* and *WorldIntegrator*), and the Operations Monitors. Since the representation of the real operation is still an ongoing research issue, the real operation is simulated by another copy of OpSim. This made testing the performance of the system very simple, since the differences between the real operation and the planned operation could be carefully controlled. This chapter will focus on the two remaining pieces of the methodology: the prototype OpSim and the prototype Operations Monitors.

B. OpSim

OpSim is an aggregate-level, discrete-event simulation in which the entities represent combat platoons and company headquarters. This prototype allowed for the reasonable representation of a friendly brigade attacking an enemy battalion. OpSim is

written entirely in Java 1.2, and the OpSim GUI makes extensive use of the Java Swing API. The basic construction of OpSim is a client-server architecture. OpSim incorporates all the requirements specified in Chapter II, except that it does not implement a standard protocol for communicating with other simulations (such as ALSP) [143]. OpSim has the ability to answer some one-time queries as well as allow clients to subscribe to information. OpSim allows the user to specify one of four clock modes, including two near-real-time modes. Two different attrition models have been incorporated into OpSim. OpSim uses *off-the-shelf* terrain. All of these aspects of OpSim are described in detail below.

1. Client-Server Architecture

The OpSim server listens on a known port for connections. When a connection is requested, a new simulation object is created as a thread (lightweight process). Multiple copies of simulation objects can be running simultaneously. When a simulation object is created, it is given two port numbers. The simulation will establish server Transport Control Protocol/Internet Protocol (TCP/IP) sockets on these two ports. The first is the port that will be used by the GUI to connect to the simulation. This is a control connection used by the GUI to give commands to the simulation, such as start, stop, pause, and resume. The GUI and all other clients use the second connection to subscribe to information. With each connection request to this port, which is a subscription request, the simulation creates a new subscription listener thread. Each client that subscribes to information from the simulation has its own connection and its own servicing thread on the server side of the simulation. The GUI, therefore, has two

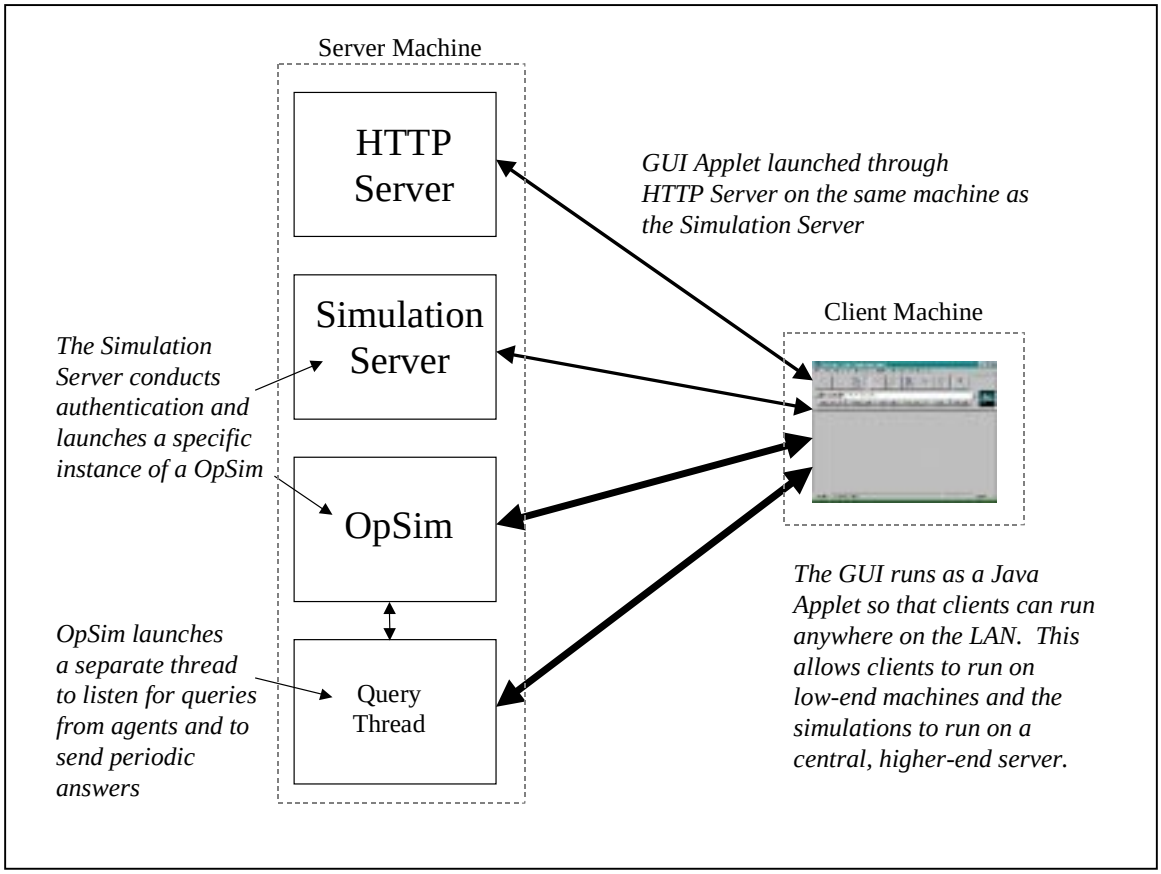


Figure 7: Connection architecture for OpSim

connections to the server – one on which to send control commands to the simulation and one on which to subscribe to information and send one-time queries.

While the simulation itself runs on a server machine (which may have significant computation power if required), the interface to the simulation is a Java Applet which can run on relatively low-end machines anywhere on the Local Area Network (LAN) and viewed in any Web browser with a Java virtual machine. As part of the Java security management scheme, Java Applets can only connect to machines from which

the applet code was downloaded. For this reason, a small HTTP Server is required to run on the same machine as the simulation server.

The connection architecture is shown in Figure 7. As will be described in Section C of this chapter, the simulation sees Operations Monitors as just another subscription connection. While these connections are routinely called "subscription connections," it should be noted that these same connections are used for the client to transmit one-time queries and for the server to reply to those queries.

This architecture was chosen so that the clients could be very thin. The GUI client only conveys commands from the various buttons and boxes on the GUI, through API calls, to the simulation, and it displays information that it has subscribed to on the screen. The movement of entities, the resolution of interactions (e.g., combat), and the management of time are resolved by the simulation on the server machine.

Control over the simulation is exercised by the user, through the GUI, which communicates to the simulation through an API. For instance, a scenario is represented in OpSim as a plan file. This plan file is critical, because it contains the list of entities and their planned actions. The plan file resides on the local (i.e., client) machine, is read by the GUI, and is transmitted to the server. This was done so that each client (human) could control the management of scenario files locally. Maintaining the plan files locally requires the GUI applet to read the file from the local file system. Normally Java Applets are not allowed to access the local file system. In order for the GUI applet to read the plan file, the GUI applet is a signed Java Applet. While the plan file resides

only on the client machine, the terrain database physically resides both at the server and client machines. This is due to the large size of the database and its static nature.

Information is passed between the client(s) and the server via a simple message passing protocol so that neither the server nor the clients are restricted to any particular language or object broker architecture. If a developer wanted to build a client in a language other than Java, the developer would only need to connect via a standard TCP/IP connection to the server and implement the message passing protocol to do so. The message protocol is defined in Appendix A.

2. Terrain Representation

Most military computer simulations today read in terrain data, translate it into a proprietary format, and then use the proprietary formatted data. This occurs due to different aspects of geographic information being in different formats, such as elevation posting being in Digital Terrain Elevation Data (DTED) format and natural and cultural features being in Digital Feature Analysis Data (DFAD) format. Since OpSim was designed for the operational environment, an involved process of converting various data sources into a unified database is impractical. For the sake of performance and usability, a better representation of terrain data for use in OpSim was needed. The National Imagery and Mapping Agency (NIMA) (formerly the Defense Mapping Agency) has a viable solution [144].

OpSim uses terrain data in Vector Product Format (VPF™ [2]). VPF is a standard format, structure, and organization for large geographic databases and is based on a relational data model. VPF allows applications to read terrain directly from computer-readable media (e.g., CD, disk, or tape) without converting it to an intermediate format. Terrain data in VPF is in vector format that concerns itself with vertices, lines, and three-dimensional coordinates. This allows for data to be accessed by spatial location and thematic content (an advantage over the raster model for terrain data) [144-146].

The specific VPF product used in OpSim is VMap-2™. VMap-2 is intended for use by tactical planners and provides terrain details at scale 1:50,000 and scale 1:100,000. OpSim uses a prototype VMap-2 database that includes data libraries for Ft. Hood, Texas, and Norfolk, Virginia. Each library contains twelve thematic coverages, such as boundaries, elevation, transportation, and vegetation [147].

NIMA has a number of standalone tools that can be used to view a variety of their mapping products. Unfortunately, none of these tools have separable libraries that can easily be used in other programs. For this reason, Haines and Surdu built a Java package to read in the data and display the map with the various coverages [8]. The existing package allows the user to specify which coverages to display over the area of interest. This work, however, has not been completed, and the capability does not exist to conduct spatial queries of the database. Since the current implementation does not

[2] VPF and VMap-2 are registered trademarks of the National Imagery and Mapping Agency.

permit a query of, for instance, elevation at a given point, line of sight computations cannot be adequately performed.

The design of OpSim envisioned the need for a file containing changes in the database [8]. This delta file would be used to record destroyed bridges, new power lines, flooded areas, etc. This would facilitate the dynamic updating of a mostly-static database. This feature has not been implemented in the prototype OpSim.

It is important to note at this point that there is no concensus on terrain format. VPF is an emerging standard, and there are a number of competitors. The Defense Modeling and Simulation Office has a terrain format, called SEDRIS. The purpose of SEDRIS was to have a uniform terrain format that people could convert their proprietary terrain to and from. If an organization wanted to conduct an exercise using both ModSAF and BBS, and the correct terrain database only existed for ModSAF, for instance, rather than paying for someone to build the BBS terrain database, the ModSAF database would be converted. The ModSAF-to-SEDRIS converter would be used, and then the SEDRIS-to-BBS converter would be used. In this way, each simulation would need to build two converters (to and from SEDRIS) rather than build a converter for every simulation with which it might likely need to interact [148]. There is some wisdom in OpSim eventually being able to read SEDRIS data as well as VPF. Why not take advantage of terrain databases that someone has already spent money to build? NIMA claims that eventually all their digital mapping products (such as DTED and DFAD) will be converted to VPF. As VPF becomes more widely accepted and terrain databases

exist for many more areas, this seems to be the most practical solution for the tactical, operational environment.

3. Query Capability and API

As mentioned earlier, there are two ways for clients to get information from OpSim. The first is to subscribe to information. Through the API, any client can subscribe to the following information: the location of entities, the status of entities, full entity updates, the simulation clock, and the simulation status (e.g., whether it is running, stopped, paused, or finished). The second method of getting information from OpSim is through one-time queries. Both types of queries are sent and the answers are received along the subscription connection, described earlier.

Recall that all clients have a subscription connection to the simulation and that when a subscription connection request is received, the simulation creates a query thread to process communication between the simulation and the client. This is shown in Figure 8. In order to facilitate the creation of new clients, three Java classes were built: QueryAPI, BasicClient, and BasicClientListener. When a BasicClient (or some object that inherits from BasicClient) is created, it instantiates a QueryAPI object and a BasicClientListener thread object. The QueryAPI class is used by a BasicClient to convert method calls (i.e. function calls) into the various, properly formatted messages (shown in Appendix A). The BasicClient does not block, waiting for a response. When the query thread gets the information it needs to answer the query (which may be an ongoing subscription for information), the query thread sends the answer back along the subscription connection. The BasicClientListener thread listens for messages on this

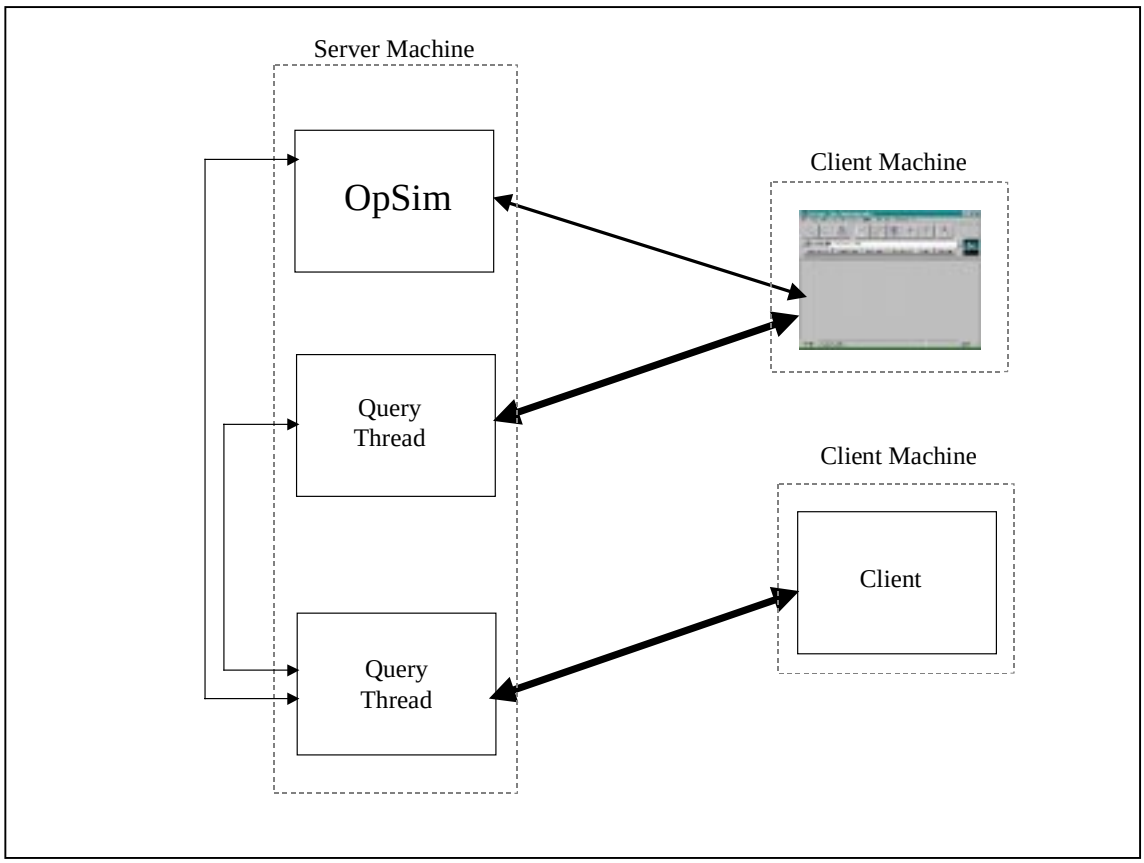


Figure 8: Two clients connected to OpSim

connection. When a message is received, the appropriate method is called to process that message. For instance, when the status update on an entity is received, the processEntityStatus(…) method is called.

To create new clients, a programmer needs to create a new client class that inherits from BasicClient ("implements" in Java parlance). Similarly, a new listener class that inherits from BasicClientListener needs to be created. Then only those methods of BasicClientListener that the programmer wants to be different from those in

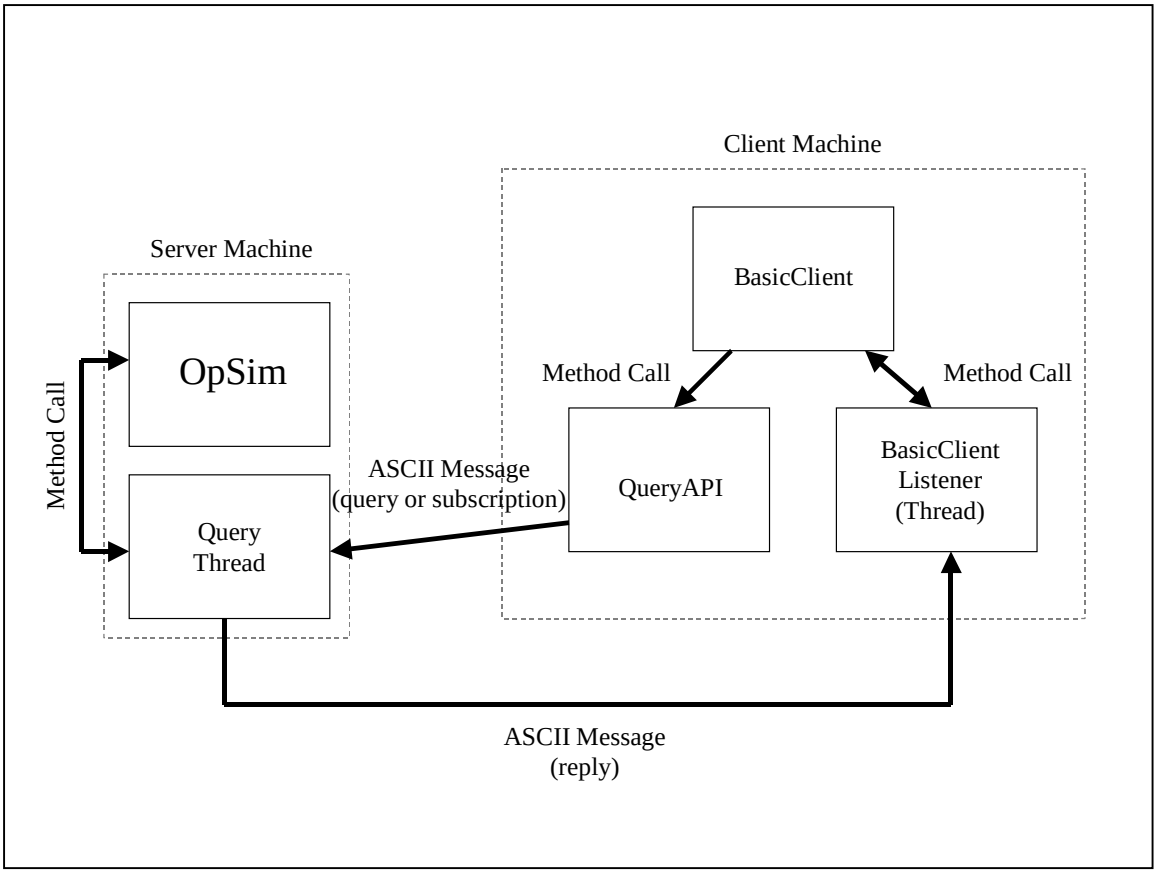


Figure 9: Interactions between BasicClient, QueryAPI, and BasicClientListener

BasicClientListener need to be rewritten. This is how the Operations Monitors (described in Section C) were created. Figure 9 shows the interaction between these objects.Since the BasicClient does not block waiting for an answer, periodically it needs to ask the BasicClient listener if the answer has been received. For subscription information, once the first answer is received, the BasicClient can just make a call to BasicClientListener to get the latest value. For one-time queries, it must do other things

until an answer is received from each query. In this way, the programmer can decide to block waiting for an answer if needed or perform other actions if not needed.

In the current implementation of OpSim, clients and messages are not prioritized. The various query threads are of equal priority, so all of the query threads try to answer queries and supply subscription information as best they can. There is, therefore, the potential for a simulation with many subscription connections to bog down.

4. Near Real Time Constraints

As discussed in Chapter II, OpSim should be able to run as fast as possible when analyzing courses of action and in near real time when the actual operation is in progress. To this end, there are four clock modes in OpSim, and the user can change the clock mode during execution. The first clock mode, "fast," allows the simulation to advance to the next event on the event queue as soon as it has finished processing an event. This is the mode used by Operations Monitors to explore the ramifications of differences between the planned operation and the current operation. It is also the mode used to analyze a variety of courses of action during planning.

When watching the simulation running in fast mode on the GUI, one sees snap shots (at the update rate set by the GUI) of whichever course of action is being executed at the time. One might see an experiment near the end of execution in one screen update and the next experiment near the beginning of its execution in the next update. This view is often counterintuitive to a viewer, but it generates information about the courses of action very quickly – which was its intent. The second clock mode, "presentation mode," was created to allow a viewer to step through the simulation very quickly, but

slowly enough that actions happen in sequence. In fast mode, a friendly brigade attacking an enemy battalion usually takes less than two seconds to simulate a two-hour operation. In presentation mode, it takes about ten minutes to view this same two-hour operation.

The last two modes are near-real-time modes. They are designed so that the simulation will be executing the plan at the same rate that the real operation is progressing. In this mode, an event will not be executed prior to its real-world time; although, it may be executed slightly later. If the server machine is very busy there is the potential in this mode for the simulation to fall behind the real operation by a significant amount. This might make the Operations Monitors detect differences between the plan and the real operation that do not really exist. For this reason, the user has the option of two near-real-time modes. The first mode, "keep," indicates that the user is willing to fall behind temporarily on the assumption that when the peak load on the machine has passed it will catch back up. The second mode, "discard," indicates that the user is willing to discard some events in the interest of remaining close to real time. Both modes have obvious drawbacks, but rather than make the choice between them as a design decision, OpSim leaves this decision to the user.

5. Plans and Plan Representation

When creating a scenario the user can create up to four plans for each of the sides involved. The number of sides allowed is set in a configuration file, and the default is six. The user can later decide which subset of all those plans to execute. For instance, the user could choose to run plans one and two of side A against plans two and four of

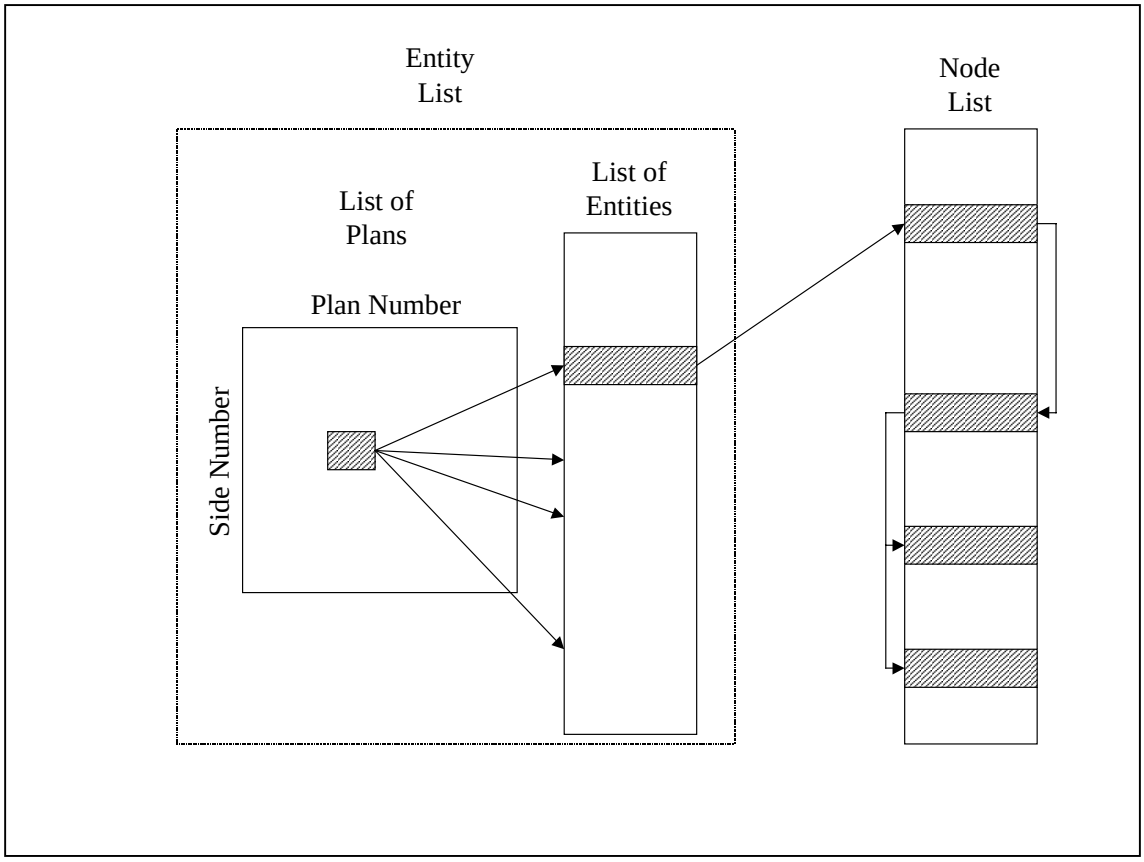


Figure 10: Relationships between plans, lists of entities, and the NodeList

side B. The user also specifies how many times each of these "match-ups" will be executed. Each execution of a match-up is called a run or experiment. In the above example, if the user specified ten iterations, the simulation would run 40 experiments.

Once the user has created the first plan, that plan can be "cloned." The cloned plan can then be modified. This facilitates the rapid creation of multiple plans. In most cases the various courses of action that a staff investigates have similar forces involved

and many of the entities will behave the same as they did in some other course of action. This cloning process helps take advantage of the similarities between plans and lets the user change only those things that are different. When a plan is cloned, copies are made of all the entities in the old plan, so each plan has its own entities. This makes it easier to delete some entities from certain plans and leave them in others. The number of plans for each side is independent. Side A might have three plans, and side B might have four.

Each plan has its own group of entities associated with it. The plans and entities are actually held in an EntityList object. The list of entities within an EntityList is implemented as a hash table using the unique identification number of each entity as its key. The plan has a list of entity identification numbers that belong to the plan. It can then use this list of identification numbers to quickly retrieve entities from the hash table. Each plan also has a series of path nodes. Each entity in a plan will move from path node to path node when the plan is executed. (This is described in greater detail in the next section.) The line segments between the path nodes are called paths. The various path nodes have unique identification numbers, and they are stored in a NodeList object, which is also implemented as a hash table. The Java HashTable class is a relatively fast implementation of hash tables, and it has a very nice series of methods that facilitate its use. The relationship between plans, lists of entities, and the NodeList are shown in Figure 10.

6. Simulation Events and the Simulation Executive

OpSim is a discrete event simulation; all events occur at an instant in time. As suggested by Evans, Wallace, and Sutherland [149], the various combat entities can

execute any of the following events: search initiation event, detection event, attack decision event, attack or defend event, known killed event, and node decision event. Each of these events is described below.

As events are executed, they may call for the creation of other events. When events are created, they are added to the simulation event queue. The event queue is implemented as a Java Vector. Events are ordered in the queue by time stamp. After an event is executed, the simulation executive asks the event queue for the next event. If the time stamp on the next event is the same as the time stamp of the event just executed, the event queue gives the event to the simulation executive. If the time stamp on the next is greater than the event just executed (it can never be less!), the action of the event queue depends on the clock mode. In "fast" mode the next event is given to the simulation executive, and the simulation clock is advanced to the time stamp of the new event. The same action occurs in "presentation" mode, but there is a half-second delay between all events. In either real time mode, the next event is returned only if the wall clock time is greater than or equal to the time stamp of the event.

Entities have plans, which govern their movement during the execution of the simulation. As the various entities move across the terrain, they must "look around" for enemy entities in order to engage them with fire. An entity "looks around" by executing a search initiation event. When executing this event, an entity checks to see if it has line of sight to any entities within 4000 meters. Entities that are farther than 4000 meters away are ignored. For any enemy entities within line of sight, a detection event is scheduled for some time in the future. This means that a new detection event is created

and added to the event queue. The time stamps on these detection events are determined by adding a random delay to the time stamp of the search initiation event. Better units have shorter average random delays.

Detection events are used to determine whether the entities "acquire" enemy targets. Just because an enemy entity is within line of sight does not mean that it will be acquired. Small variations in the ground are generally not reflected in digital terrain databases, and often these small dips in the ground are sufficient to hide men and machines. Entities at maximum visibility range are harder to acquire than those at short ranges. Entities in buildings, woods, or other broken ground are difficult to acquire. Also, the personnel represented by the entity may be busy fighting with another unit. They might also be looking in the wrong direction. For these reasons, a probability of acquiring the enemy entity is computed, and a random uniform variate is generated. If this "die roll" is less than the probability of acquiring the target, an attack decision event is generated for some time in the future. As mentioned earlier, the use of VPF terrain is not fully implemented, so this computation of acquisition probability is not very realistic.

Attack decision events are used by entities to decide whether they will begin to fire at an enemy entity. This decision is based on what type of entity the enemy entity is and what weapons the friendly entity has. For instance if an entity has no anti-tank weapons that can reach the target, it does not make sense to engage the target. If the entity decides to attack the enemy entity, an attack event is created for some time in the future.

Attack events are used to resolve combat. When an attack event is executed, both the attacking entity and the defending entity stop moving. They will remain in place until one or the other has been destroyed. Then the surviving entity will continue to move along its planned route. An attack event uses a series of calls to an attrition model to determine the effects of combat, such as number of men lost, number of tanks lost, and the number of weapons lost. The "duration" of an attack event is determined stochastically. The attrition models determine a casualty rate based on strength differentials. The number of casualties is determined by applying this rate of casualties over time. Casualties are assigned to the entities, and their states are updated. If after the attack event is resolved, neither side is destroyed, another attack event is scheduled for a time in the future equal to the current time plus the duration of the current attack event.

It would be possible to compute the casualty rate for the attacker and defender and determine in one step when one or the other entity would be destroyed. The reason that combats are resolved in smaller intervals is to allow other entities to participate. As other entities' search initiation, detection, and attack decision events are executed, they may begin to fire on entities that are already engaged. This allows a larger force to potentially bring more of its units to bear. Note, however, that any single entity will only schedule attack events against one other enemy entity. This was done so that an entity could not "act" like multiple entities by being involved in several fights at full strength before casualties are resolved.

The process of scheduling attack events continues until one of the two entities is destroyed. When one entity is destroyed, it schedules a known killed event for some time in the future. Until this known killed event is executed, enemy entities think it is still alive and may spend time firing at the unit. This is an accurate reflection of combat situations. Once the known killed event is executed the other entity begins to move again. It does this by scheduling a node decision event. The time stamp on the node decision event is determined by the entity computing the time to travel to its next plan node.

Node decision events are used for entities to choose between one of several divergent plan paths. As described in the plan representation section more than one path my lead out of any given node (but only one can lead into a node). When an entity executes a node decision event, it chooses between one of the possible paths and schedules another node decision event for a time when it should reach the next node in its chosen path. Each of the paths leaving a plan node can have one or more conditions associated with it. OpSim assumes that the departing paths had been created in order of decreasing priority, so the entity will decide to move on the first path for which all conditions are met. In the prototype OpSim, the ability to assign conditions to a node was not implemented; although, all the hooks are in place. If a path has no conditions associated with it, the entity assumes that all conditions are met, and it chooses that path. For this reason, the first path entered from each plan node is the one the entity will choose.

There are three conditions at path nodes for which hooks have been left in the OpSim code. The first is a no-earlier-than (NET) condition. This means that the entity cannot travel along that path any earlier than the time associated with the condition. The second is a no-later-than (NLT) condition. This means that the entity cannot move along the path after the specified time. This would be useful if an artillery-delivered minefield was going to be dropped along that path at a certain time, for instance. The third is a strength condition. This specifies the minimum strength an entity must have to move along the path. Even if there is only one outgoing path, that path can have conditions associated with it. A path can have multiple conditions. There is also a need for a condition that bases the node decision on other entities. For instance, a planner might specify that an entity will not move along a path until some other entity has reached some location.

Throughout this discussion of events, there has been little mention of any reasoning that the entities perform as they attempt to accomplish their missions. The entities were intentionally created with little reasoning capability. Decisions that need to be made in terms of attack decision and node decisions are well understood and (except for the creation of random variates) algorithmic. This was done so that combat effects that are well understood would cause any differences between the plan and the current operation. The differences would not be caused by decisions made by the entities that might not be in accordance with the commander's intent. Entities will follow their plans exactly as defined by the user.

The simulation executive for OpSim is very simple. Each entity and each simulation event is an object. The simulation executive calls the simulation event queue to get the next event to execute. A simulation event has a pointer (a handle in Java parlance) to the entity or entities involved in the event. When the simulation executive receives an event, it increments the simulation time to the time of the event. Then the simulation executive tells every entity to execute a move. The entities do this by moving the appropriate distance indicated by the current time, the time of its last move, and its movement rate. After all entities have moved, each one of them also executes a search initiation event to attempt to see any new entities that have come into view based on this round of movement.

Each simulation event object has an executeEvent(…) method. The simulation executive calls this method and waits for the return value. Then it either waits for half a second (in "presentation" mode) or immediately asks the simulation event queue for the next event. A simulation experiment ends when one of two conditions are met:

- All the entities on one side have been destroyed or
- All the surviving entities on both sides have reached the end of their plans and no activities are taking place.

The user defines how many experiments OpSim is supposed to run. When an experiment ends, if OpSim needs to execute more experiments, it resets the simulation, puts some initial search initiation events on the simulation event queue, and begins executing the next experiment.

Whenever the simulation event queue has fewer than five events on the queue, the simulation executive adds some search initiation events at random times into the future. OpSim is a discrete event simulation, and it skips long periods of inactivity by incrementing the clock to the next event. Entities move in "jumps," however, and if the gaps between events executing are too great, two entities may pass each other without having an opportunity to detect and engage. This mechanism of adding search initiation events is also used to initialize the simulation when a new experiment begins.

7. Attrition Modeling

In Chapter II a number of attrition models as well as issues related to attrition models were described. It was not the purpose of this research to propose a new and better attrition model. A goal of the OpSim design was to make OpSim flexible enough that it did not depend on any particular attrition model. To this end, an abstract class, AttritionModel, was created, and two specific attrition models that inherited from AttritionModel were built. The first was the Dupuy QJM model [51, 52]. The second was an *ad hoc* model based on recreational war games. The QJM model is a deterministic model, and the basic model was modified slightly to make it stochastic. In addition, the QJM model was designed for large forces (e.g., divisions and corps), so some modifications were made in order to apply it to platoons.

The AttritionModel class specifies methods that compute the casualty rates of the two forces. Given a specified duration of the fight, a series of other methods computes the actual number of men, armored personnel carriers, tanks, wheeled vehicles, air defense weapons, anti-tank weapons, and infantry heavy weapons that are destroyed

within that time interval. These losses are initially stored in temporary variables in each entity. Once all combats have occurred for that time cycle (all attack events with the same time stamp) are executed, all entities that participated in attacks update their permanent variables. All events with the same time stamp are assumed to happen simultaneously in a discrete event simulation. This process ensures that all entities have the same strength at the beginning of all attack events with the same time stamp.

In OpSim, entities have a rating of their effectiveness, called the Combat Equivalence Value or CEV. OpSim interprets this CEV as the mean performance of the entity. The probability distribution associated with CEV is stored in the entity as well. In the current implementation, CEV can be a parameter of either a Normal or Exponential distribution. For distributions that require a second argument, each entity also has a CEVSigma parameter. When entities participate in attack events, random numbers are generated according to the specified distribution and parameter(s). These numbers are used to modify the combat results. For instance, an M-1 tank platoon has a Normal distribution with CEV of 1.2 and a CEVSigma of 0.2. In the two implemented attrition models (the *ad hoc* and the QJM) a random variate is generated according to this distributions, and that number is multiplied by the strength of the unit to determine its final strength.

8. Performance

The OpSim prototype has capabilities that focus on the planning and execution phases of an operation. As the planners develop several friendly and enemy courses of action, these can be created in OpSim as described previously. The user can then choose

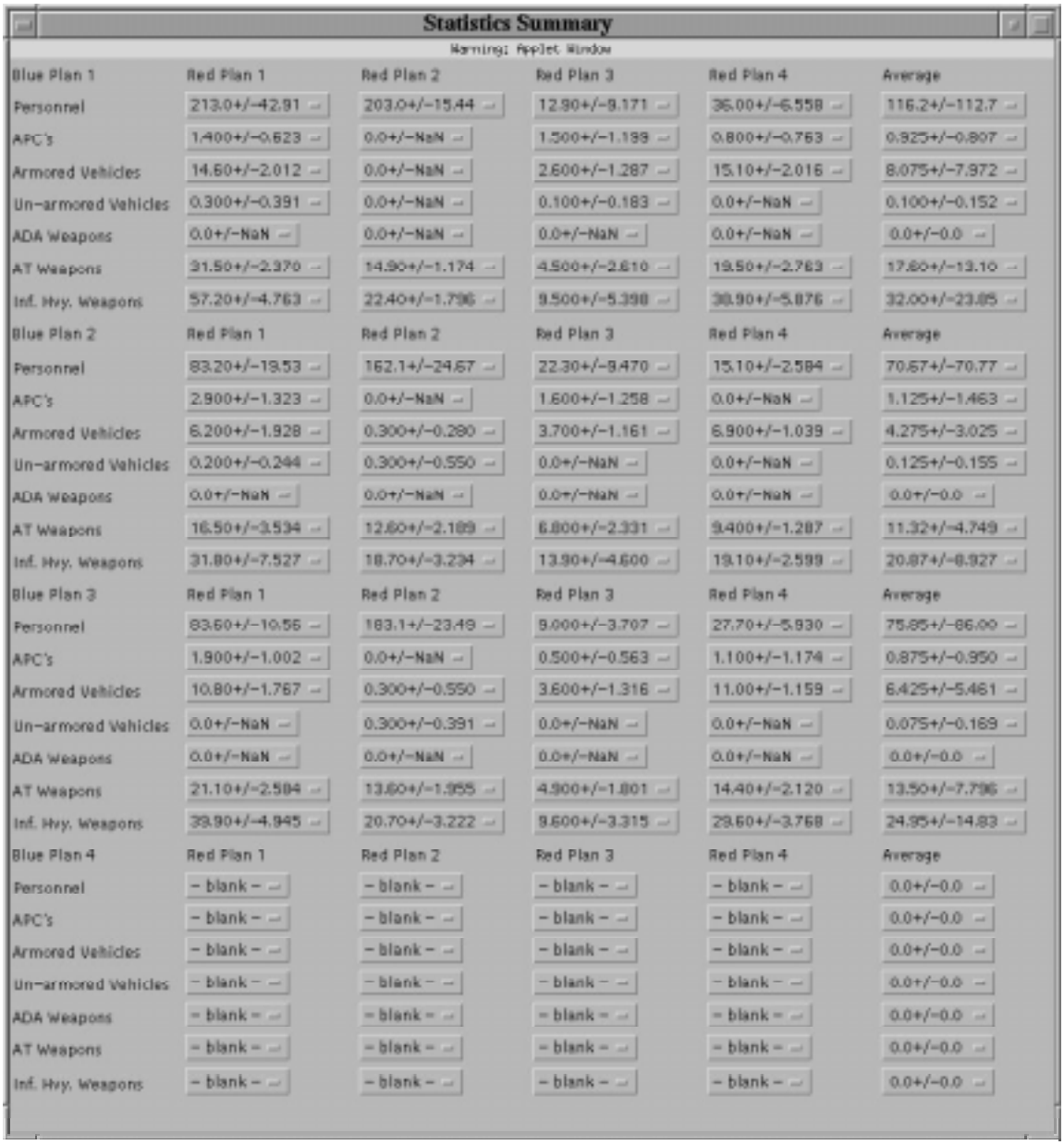


Figure 11: Summary statistics created by OpSim

which (some or all) of the courses of action for each side to simulate. The user also chooses how many iterations of each "plan versus plan" to run. On a Sun Ultra 5

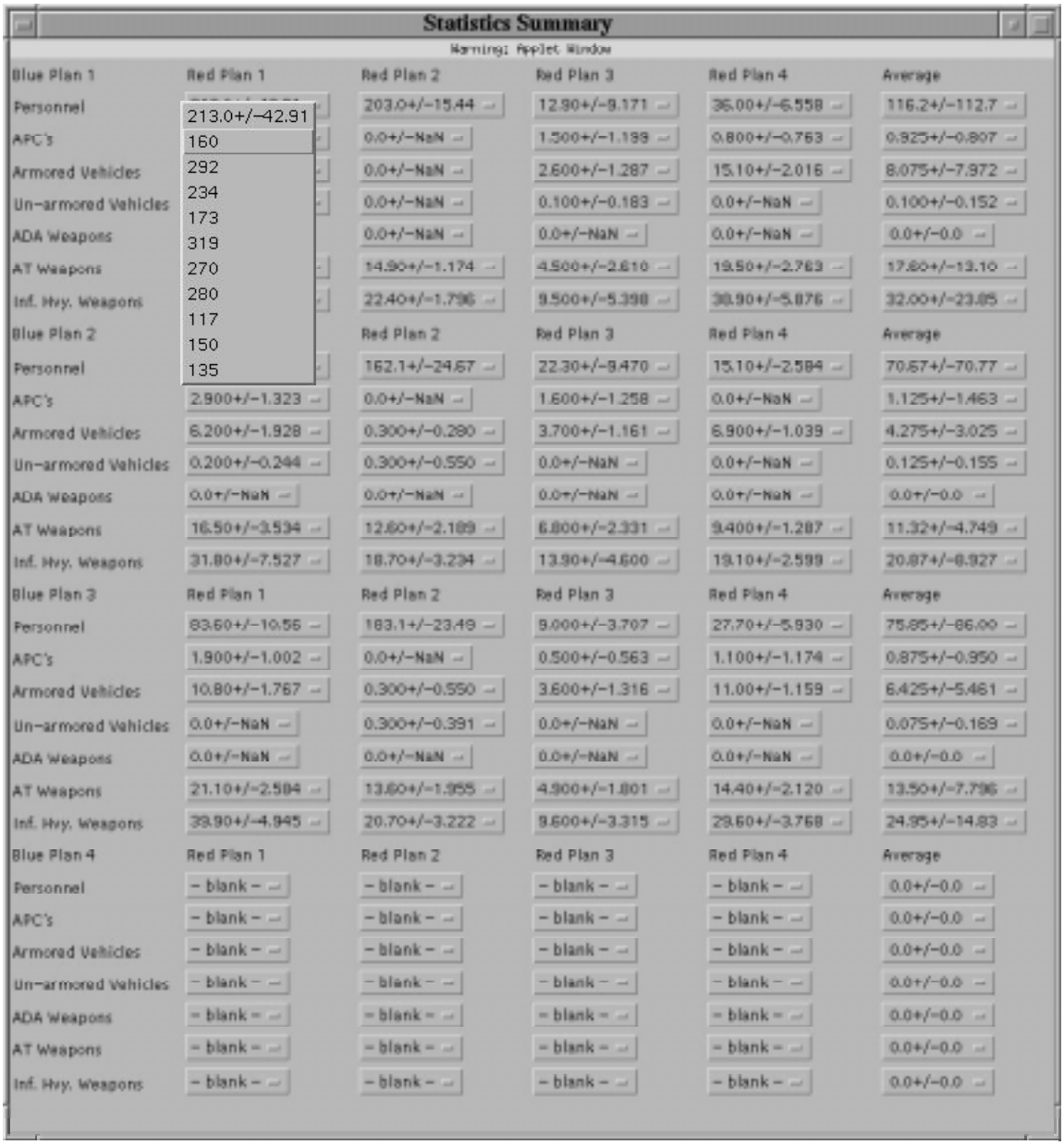


Figure 12: Personnel losses expanded to show the results of each experiment

workstation, each iteration of a brigade attack on an enemy battalion takes approximately two seconds. So with two sides, having four plans each, for ten

iterations, it takes OpSim approximately 320 seconds – just over five minutes. Once the simulation has run all these experiments, a window is created which summarizes the statistics for all the selected iterations. This is shown in Figure 11.

Each entity has some number of personnel, armored vehicles (tanks), armored personnel carriers, unarmored vehicles, air defense weapons, anti-tank weapons, and infantry heavy weapons (e.g., machineguns, small mortars, etc.). During the execution of an experiment, entities report losses to a statistics collector object. After all the experiments are done, the statistics collector computes means and confidence intervals around those means of the losses each side took. For instance in Figure 11, Side A, "Good Guys" lost an average of 213.0 +/- 42.91 personnel when running plan 1 versus Side B's plan 1. The confidence interval is important, because it gives the user some indication of the variability of the results of the individual experiments. Each of the cells in the display is implemented as a list box, so if the user clicks on the cell, the losses from each experiment are shown, as depicted in Figure 12.

Along the right column of the statistics window are the summaries across all enemy plans. These are the means of the means, so to speak. This gives some indication to the user how well a friendly course of action performs across all likely enemy courses of action. Again, by selecting on a cell, the data used to compute the cell value are shown.

The intent of this statistics window is to provide some data that could be used along with other decision criteria in selecting a course of action for the upcoming operation. Obviously casualties are not the only factor in choosing a plan. There are

several lists of factors that are often used to evaluate plans. Among these are the Principles of War, Tenets of Army Operations, and the elements of combat power [142]. Staffs generally use tools, called decision support matrices [63], to help evaluate alternative courses of action, and the statistics generated by OpSim might be used as several of the evaluation criteria.

OpSim is written in Java, which is an interpreted language. Interpreted languages are generally slower than compiled languages; however, since the Java standard has stabilized at version 1.2, the developers have had time to begin to optimize the Java compilers. In addition, just-in-time compilers are becoming available, many for free, which compile the Java byte codes into machine-specific binaries during execution. In addition, since OpSim was written as a prototype, there was little emphasis on performance optimization. Despite the slow speed of OpSim relative to how fast it *might* be after some compiler and code optimizations, the current implementation is fast enough to allow staffs to quickly evaluate many courses of action and also experiment with some branches and sequels. The analysis done by OpSim is less susceptible to personal bias, group think, and the other issues discussed in Chapter II. In a production system, *the planning capabilities provided by OpSim would greatly enhance the ability of staffs and commanders to make rapid tactical decisions*.

There is one shortcoming to the statistical analysis done by OpSim. While the statistics collector can handle as many sides and plans as OpSim can, the statistics display window can only display two sides at a time. Only Side 1 (usually the friendly forces) and Side 2 (usually the enemy forces) are displayed. OpSim also assumes at this

point that all sides other than one's own are the enemy, so the user cannot very well specify coalition types of operations. These problems could be remedied without much difficulty, and the proposed solutions will be discussed in Chapter V.

Once the commander has chosen one of the courses of action to execute, OpSim can then be used to monitor the current operation. OpSim incorporates a number of capabilities that facilitate this use. As described earlier, any number of clients can connect to the simulation and subscribe to information. The user can set the clock to one of the real-time modes. In one of the real-time modes, the plan (simulation) should progress at the same rate as the real operation. If the real operations is falling behind or getting ahead of the plan, the clients can get this information from the simulation (and the representation of the real operation). Another useful feature of OpSim is some user-level control over the generation of random variables.

There are two random number generation modes. OpSim wrapped the Java Random class in another class, OurRandom. This was done so that random number generators other than the one provided in the Java programming language could be easily implemented without modifying any other code in the program. OurRandom has two modes that can be changed during execution, if desired. The first mode, "random," generates pseudo random numbers according to the distribution parameters provided. The second mode, "expected value," generates a number that is equal to the expected value of the distribution with the specified parameters. While running many experiments during planning, the random mode should be used. While running the simulation in parallel with the real operation, the expected-value mode should be used.

Expected-value mode should be used, because external clients should not decide that the plan is going awry due to a series of unlucky "die rolls." Expected-value mode, like the lack of reasoning in entities is used as a variance-reduction technique.

C. Operations Monitors

1. Overall Structure

The heart of the proposed methodology is the use of software agents, or Operations Monitors, to compare what is happening in the real operation with what was supposed to happen according to the plan. As described previously, the Operations Monitors exist in a tree-like hierarchy. The Top Level Monitor (described below) is the first to be created. All subsequent monitors are created as children of another monitor. All of the monitors in the prototype implementation are threads running in the same Java virtual machine. OMs communicate with child OMs by calling methods and with parents by calling the parent's report() method.

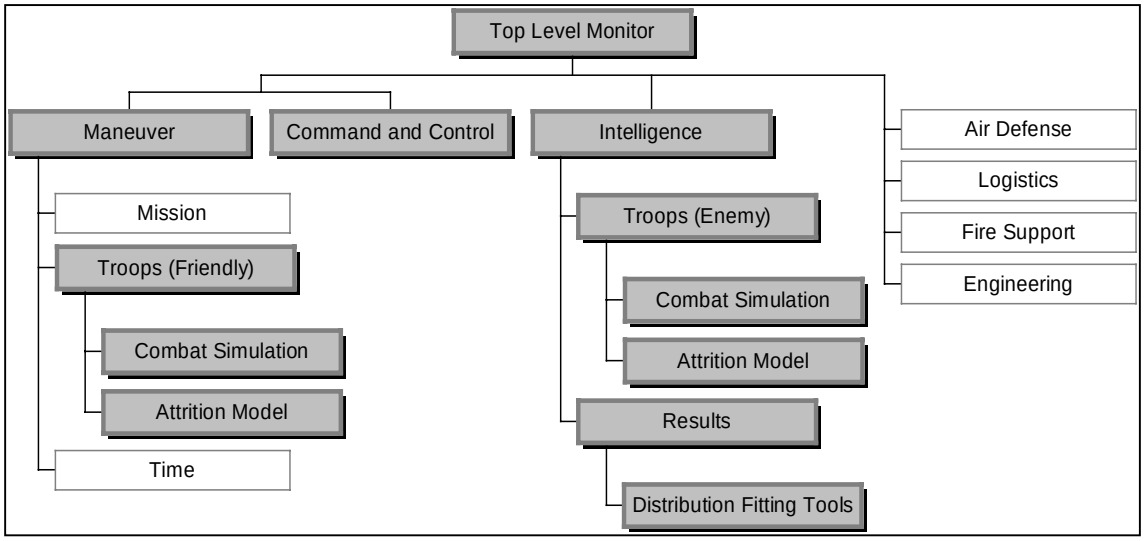


Figure 13: Implemented Operations Monitors (in dark boxes)

2. Implemented Monitors

Each monitor is responsible for a very narrow domain. In Chapter III the basic policy for determining the default initial set of monitors was described. This was based on the unit's mission. The monitors that could be implemented were based on the kinds of information that the prototype OpSim could provide. The seize mission described in Chapter III and the current state of OpSim determined which Operations Monitors to implement. Figure 13 shows in gray which monitors were implemented.

All OMs inherit from the BasicMonitor class. BasicMonitor handles the creation, management, and deletion of child monitors. Each BasicMonitor has a subscription connection to the real and simulated worlds. The BasicMonitor class manages the creation of these connections as well as the passage of queries,

subscriptions, and answers. The BasicMonitor class manages the movement of messages up and down the tree of OMs. Finally the BasicMonitor class contains the basic control loop of any OM.

Periodically OMs wake, do some work, make some decisions, and go back to sleep. When they wake, three methods are called in sequence: classifyInformation(), makeSomeDecisions(), and getNextBatchOfInformation(). The BasicMonitor class calls each of these methods. In BasicMonitor, these methods are empty. The developer of a new OM must fill in the details of these methods. For instance, in the Forces Monitor, the makeSomeDecisions() method uses three fuzzy rules bases and a crisp rule base. In the TopLevelMonitor, the makeSomeDecisions() method computes the value of a utility function. By implementing these three methods, the developer controls the logic of any new OM.

At the first level of the dynamic tree structure of OMs, the level that corresponds to Battlefield Functions [142], the Maneuver, Command and Control, and Intelligence monitors were implemented. Since OpSim could not stimulate the Air Defense, Logistics, Fire Support, or Engineering Battlefield Functions, these OMs are merely stubs. The Maneuver, Command and Control, and Intelligence OMs report to the Top Level Monitor. The Top Level Monitor computes a utility function based on the reports and decides whether to take some action(s). In the remainder of this section, each of the implemented monitors will be described. The discussion will begin at the leaves of the tree and end with the Top Level Monitor.

3. How the Results Monitor Works

The Results Monitor is designed to compare the performance of types of entities in the simulation with their real-world performance. Recall that the performance of entities in OpSim is described by some probability distribution (e.g., Normal or Exponential). The results of various combat actions by entities are archived by class. M-1 equipped tank platoons have one set of results; T-72 equipped platoons have a second set; and so on. On the "real" side, the assumption is that outcomes of combats can be used to extract the various "die rolls" that would have produced those results. The Results Monitor then compares the two sets of data for each entity type.

Both Chi-Square and Kolmogorov-Smirnov (KS) goodness of fit tests were implemented. The KS test is the default. If the KS test determines that the distribution described by the simulation's results does not match those of the real operation (within a 95% confidence), the Results Monitor attempts to determine the correct distribution. It begins by using Maximum Likelihood Estimators (MLE) [141] to determine probability distributions which might match the real data. Using the parameters determined by MLE, sets of pseudo random numbers are generated for these various distributions. The KS test is then used to determine which distribution most closely fits the real data. The closest distribution is recommended. If the closest distribution is within the same family as the current simulation distribution (e.g., Normal) this recommendation is sent to the Results Monitor's parent as a CONCERN message. If the recommended distribution is from a different family, this recommendation is wrapped in a WARNING message.

4. How the Combat Simulation and Attrition Monitors Work

When other monitors (dsecribed below) determine that the effects of some difference between the plan and the real operation must be explored, the monitors can launch either an Attrition Monitor or a Simulation Monitor. The purpose of these monitors is to project the current state of the real world into the future to determine the end state of the operation. Then the projected end state of the real operation can be compared to the estimated end state of the plan. This comparison allows the OM that launched the Simulation or Attrition Monitor to decide whether the current difference between the plan and the real operation has a significant impact on the accomplishment of the mission.

The differences between a Simulation Monitor and an Attrition Monitor deal with accuracy and time complexity. The Simulation Monitor gives a better answer, but it takes much longer to get an answer than an Attrition Monitor. The calling OM can decide which to use based on the current resource constraints on the system (i.e., if the system is busy, the calling OM might decide to use the Attrition Monitor).

The Simulation Monitor is merely a BasicMonitor wrapped around an OpSim. When a Simulation Monitor is created, the OM that created it passes in an Entity List and Node List as well as some other parameters. Normally, these parameters indicate that the Simulation Monitor should run the current state of the real operation to the end of the operation, but a Simulation Monitor could also be used to conduct “what-if” analyses. The Simulation Monitor runs the required experiments and then tells the

calling OM that it has finished. The calling OM can then query the Simulation Monitor for information, such as the projected losses and time to complete the operation.

The Attrition Monitor is an attrition model wrapped in a BasicMonitor. To project the current state of the real operation to end state, the Attrition Monitor needs to know the duration of the operation. As an estimate of this, the Attrition Monitor uses the planned length of the operation. All units on all sides are aggregated, and one large combat is resolved with duration equal to the time remaining in the planned operation. Based on the attrition model used the Attrition Monitor estimates the losses on all sides. Clearly this method will be both faster and less accurate than using a simulation; however, the calling monitor may decide that speed is more important than accuracy in certain circumstances. Another drawback of the Attrition Monitor is that it cannot be used to estimate how long the operation will take to complete.

5. How the Forces Monitor Works

A Forces Monitor is used to compare the strength of a designated force in the plan with the strength of that same force in the real operation. A Forces Monitor only looks at one force (side). When a Forces Monitor is created, the OM that created it tells the Forces Monitor which side to watch. In the test scenarios, only two sides were used, so there are generally two Forces Monitor, a Red Forces Monitor (enemy) and a Blue Forces Monitor (friendly). Regardless of which force the Forces Monitor is watching, it performs the same actions in the same manner.

The Forces Monitor has the most sophisticated reasoning mechanisms of any of the prototype monitors. It uses three fuzzy rules bases as well as a crisp rule base. The

fuzzy rule bases are used to classify differences, and the crisp rule base is used to control the actions of the monitor. The fuzzy rule bases were built with the MATLAB Fuzzy Toolbox. MATLAB facilitates the rapid creation, display, and editing of fuzzy rules. The output from MATLAB is an FIS file. A Java implementation of the Standard Additive Model (SAM) of resolving fuzzy rules was created as part of the OpSim development effort [86]. This implementation of SAM works for trapezoidal, triangular, and Gaussian membership functions. When the Forces Monitor is created, it makes

Table 4: Sample fuzzy rules

Sample Number Classifier rule:
If (percentDifference = manyFewer) &
(simulatedNumber = veryMany) &
(realNumber) = veryMany
Then
(criticalityOfDifference = significant)

Sample Criticality Classifier rule:
If (baseCriticality [from NumberClassifier] = significant) &
(percentOfTotal [of that particular type of unit] = high)
Then (criticalityClass = high)

Sample Effects Classifier rule:
If (percentTotal = medium) &
(percentChange = medium)
Then (effect = concern)

The first two rules (from the first two fuzzy rule bases) are used by the Forces Monitor to infer whether it is necessary to launch a Simulation Monitor or an Attrition Monitor to determine the probable effects of the differences. The last rule (from the third fuzzy rule base) is used classify any differences predicted by the Simulation or Attrition Monitor.

three FuzzyDatabase objects and loads each one with the information from an FIS file.

The first two FIS files are used to classify the differences between the real operation and the plan. The Forces Monitor computes the number of infantry units, tank units, etc. These computations take into account the strength of the units and their combat equivalency value (CEV). Factors used by the fuzzy rules to determine the criticality of any differences are the percent change from the plan to the real operation, the actual number in the simulation, the actual number in the real operation, and the importance of that type of platoon to the overall force. (Sample rules are shown in Table 4, and the complete rule bases are in Appendix B.) For instance, losing one tank platoon does not sound like much unless there was only one tank platoon. Also a small percent change in the number of infantry platoons lost might mean several hundred soldiers if there were many infantry platoons. The output from the two fuzzy rule bases is a measure of the criticality of the difference between the real operation and the plan. If this criticality exceeds some threshold, the Forces Monitor chooses to execute either a Simulation Monitor or an Attrition Monitor.

When either the Simulation Monitor or the Attrition Monitor finishes, the third fuzzy rule base is used to categorize the difference between the end state casualties in the plan and those computed by a child Simulation Monitor or Attrition Monitor. This categorization is done by equipment type (e.g., personnel, armored personnel carriers, tanks, etc.). The fuzzy rule base uses the percent of losses and the percent change in losses to make this determination. For instance, the change in the real operation might

double the number of armored personnel carriers lost, but these doubled losses might still be a small percentage of the total number available.

The analyses done by all three fuzzy rule bases assume that the losses for the currently executing friendly course of action versus the currently executing enemy course of action were acceptable during planning. If this were not true, why would the commander and staff choose that course of action? For this reason, the fuzzy rule bases always compare new estimates (such as personnel losses predicted at end state) with those determined during the planning process. The losses estimated during the planning

Table 5: Sample crisp rules

Sample rules (in pseudo code) used by a Forces Monitor to determine whether to launch a Simulation Monitor or an Attrition Monitor, based on inferences made by the first two fuzzy rule bases:

```
If   (combined criticality > some threshold) &
     (available resources are medium or high)
Then (launch as Simulation Monitor)

If   (combined criticality > some threshold) &
     (available resources are low)
Then (launch an Attrition Monitor)

If   (combined criticality < some threshold)
Then (do nothing)
```

Sample rule used by a Forces Monitor to report the results of running a Simulation Monitor or an Attrition Monitor:

```
If   (concernThreshold < criticality of difference < warningThreshold)
Then (report that strength of side X has a moderate impact on friendly casualties)
```

process were summarized in the statistics table, shown in Figures 11 and 12.

The crisp rule base in the Forces Monitor was written in the CLIPS language [83]. JESS (the Java Expert System Shell) was used to execute the rules. JESS is a Java implementation of CLIPS with significantly greater flexibility and power. Only the basic functions of JESS were used in the Forces Monitors. The Forces Monitor creates a Rete object, and the Rete object reads the CLIPS rules from a file. The Forces Monitor uses the Rete API to assert facts into the rule base, control execution of the rules, and exchange information with the Rete object through Store and Fetch operations. When the JESS rules want the Forces Monitor to report some information, the report is placed in a REPORT variable by JESS using a STORE operation. Periodically the Forces Monitor polls the REPORT variable using a FETCH operation. When the Forces Monitor finds something in the REPORT variable, that report is sent to the Forces Monitor's parent. Table 5 shows some sample crisp rules used by the Forces Monitor.

6. How the Command and Control Monitor Works

The job of the Command and Control Monitor is to compare the command and control nodes of all sides in the plan and the real operation. If the majority of the command and control nodes are destroyed or damaged, it will be very difficult for the maneuver entities (e.g., tank and infantry platoons) to accomplish their missions. For purposes of this prototype, command and control nodes are company headquarters elements. The various network and signal nodes that compose the communications infrastructure are not represented in OpSim, so there is no way for the Command and Control Monitor to measure them.

The Command and Control Monitor periodically counts the number of command and control nodes that are green, yellow, red, and black (colors which represent the status of units in OpSim, green being the best). As described in the discussion of sides in OpSim, OpSim assumes that all sides other than one's own are enemy. The Command and Control Monitor makes the same assumption. When the status of friendly command and control nodes is worse than in the plan, the Command and Control Monitor sends an adverse report to its parent OM. Similarly when the status of enemy command and control nodes is better than in the plan, the Command and Control Monitor send adverse reports.

7. How the Top Level Monitor Works

The Top Level Monitor collects the reports from all of its child OMs and makes decisions about whether to advise the commander of problems. The Top Level Monitor has three status levels. The Good status indicates that the operation is going better than the plan or only slightly worse than the plan. The Concern status indicates that conditions in the real operation are worse than in the plan (or will be at end state), but the overall success of the mission is not at risk. The Warning status indicates that the success of the mission is at risk. When the Top Level Monitor is in Concern or Warning status, the Top Level Monitor creates a window on the screen that describes all of the factors that contributed to this adverse status. In Good status, no reports are made.

The Top Level Monitor determines its state by computing a utility function based on any reports it has received. Good reports have positive values, and bad reports have negative values. When the value of the function drops below a definable threshold, the

Top Level Monitor is in Concern status. When the value of the function drops below a second threshold, the Top Level Monitor is in Warning status.

The number that is added to the utility function is based on the severity of the bad news. Recall that lower-level OMs attach Warning and Concern tags to their adverse reports. This information is used to determine how bad the bad news is. The Top Level Monitor determines the value associated with good and bad reports as it adds them to the utility calculation.

The Top Level Monitor also decides whether suggestions provided by a Results Monitor require user intervention. The decision is based on the type of update suggested by the Results Monitor. If the Results Monitor suggests that the family of distribution describing some entity is correct, but the parameters need to be adjusted, the Top Level Monitor just tells the simulation to make the changes. If on the other hand the Results Monitor suggests that the family of distribution must be changed, the Top Level Monitor asks the user to confirm the decision. Many families of distributions are commonly used for certain types of problems. It might not make sense to represent the performance of an entity as a Weibull or Gamma distribution.

The implementation of the Top Level Monitor was intentionally kept simple. This simplicity was designed to show that useful behavior could be accomplished without large overhead. Since many of the lower-level monitors make inferences and decisions within their narrow domain, the Top Level Monitor's work is simpler. Eventually a more intricate and more robust reasoning mechanism would be needed. Some inferences about the risk posed by the current state of the operation on the

eventual outcome are probably more difficult. It seems that some bad news should be added to the utility function, but sometimes the effect of two pieces of bad news are greater than their sums. This more sophisticated reasoning system has not yet been designed or implemented.

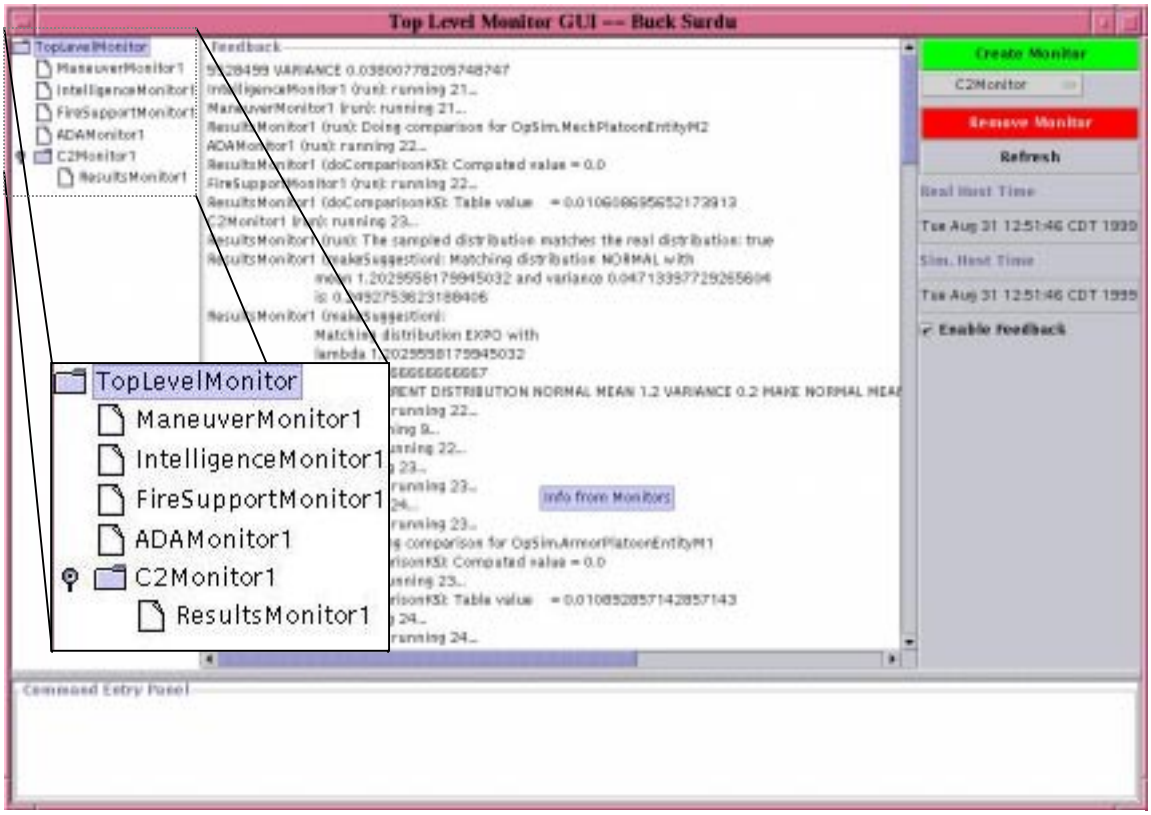


Figure 14: The Top Level Monitor GUI

For purposes of this prototype, the Top Level Monitor also owns the GUI associated with the dynamic tree of monitors. From this GUI, the user can manually launch other monitors or kill monitors that are running. Killing OMs from which parent

OMs are waiting for reports can cause the parent OMs to block indefinitely, so deleting monitors must be done with care. More importantly the GUI provides a tree-like depiction of the currently running OMs. In addition, the text window, shown in Figure 14, provides an area for the Top Level Monitor to report information to the user.

D. Summary

The implemented system involves a discrete event combat simulation (OpSim) and a dynamic hierarchy of software agents (Operations Monitors). OpSim is used to simulate plans and provide feedback to the user about expected losses. The Operations Monitors (OMs) can subscribe to information from OpSim so that they can compare the progress of the real operation to that of the plan. The prototype OMs use a combination of fuzzy rule-bases, crisp rule-bases, and utility functions to determine if the success of the plan is at risk. The OMs can also launch other, child OMs to conduct further analysis or run simulations. The top-level OM analyzes the reports from the various OMs below it in the hierarchy, computing a utility function. If the top-level OM determines that the plan is at risk, it advises the decision-maker.

CHAPTER V

RESULTS

A. Introduction

This chapter describes the results of this research. Two domain experts were shown the implemented prototype system, and their feedback was used to evaluate the worth of the overall methodology, the usefulness of the prototype, improve the prototype, or identify future work. The process used is described below, as well as the results.

B. Comparison to Other Systems

There are no existing systems with these capabilities. As mentioned in Chapter II, there are numerous Department of Defense and Department of the Army combat simulations. These simulations are unsuitable for this application for a number of reasons: large exercise set up time and cost, large numbers of required workstations required, and large number of personnel required. For these reasons the prototype OpSim was developed. The prototype OpSim incorporated all the capabilities needed in an operationally focused simulation [8].

Given the existence of an appropriate simulation, however, there are still no systems designed to run a simulation of the operation in parallel with the real operation in the military domain. While Davis and his collaborators have applied a similar technique to flexible manufacturing systems [22-25, 33], it has not previously been applied to military operations. All of the analysis of how well a plan is working is still done by humans. Since there are no existing systems with which to compare the methodology proposed in this research, an alternate method of evaluation was used.

C. Experiments

As discussed in Chapter IV, a prototype system was built to demonstrate the feasibility of the methodology proposed in this research. Once the prototype was completed, several experiments were conducted with the purpose of verification and validation. Since the WorldView portion of the methodology is still an open research issue, a second copy of OpSim was used to represent the real operation. The parameters and/or entities in the "real" OpSim were modified in specific, controlled ways to perform the various experiments.

In all these experiments, the attacking force consisted of a friendly brigade, and the defending force consisted of an enemy battalion. The entities were a mix of infantry, mechanized infantry, and tank platoons. Company headquarters elements were represented in the scenarios.

The first set of experiments involved testing the adaptive nature of the overall simulation. In the "real" OpSim, the parameters that described the probability

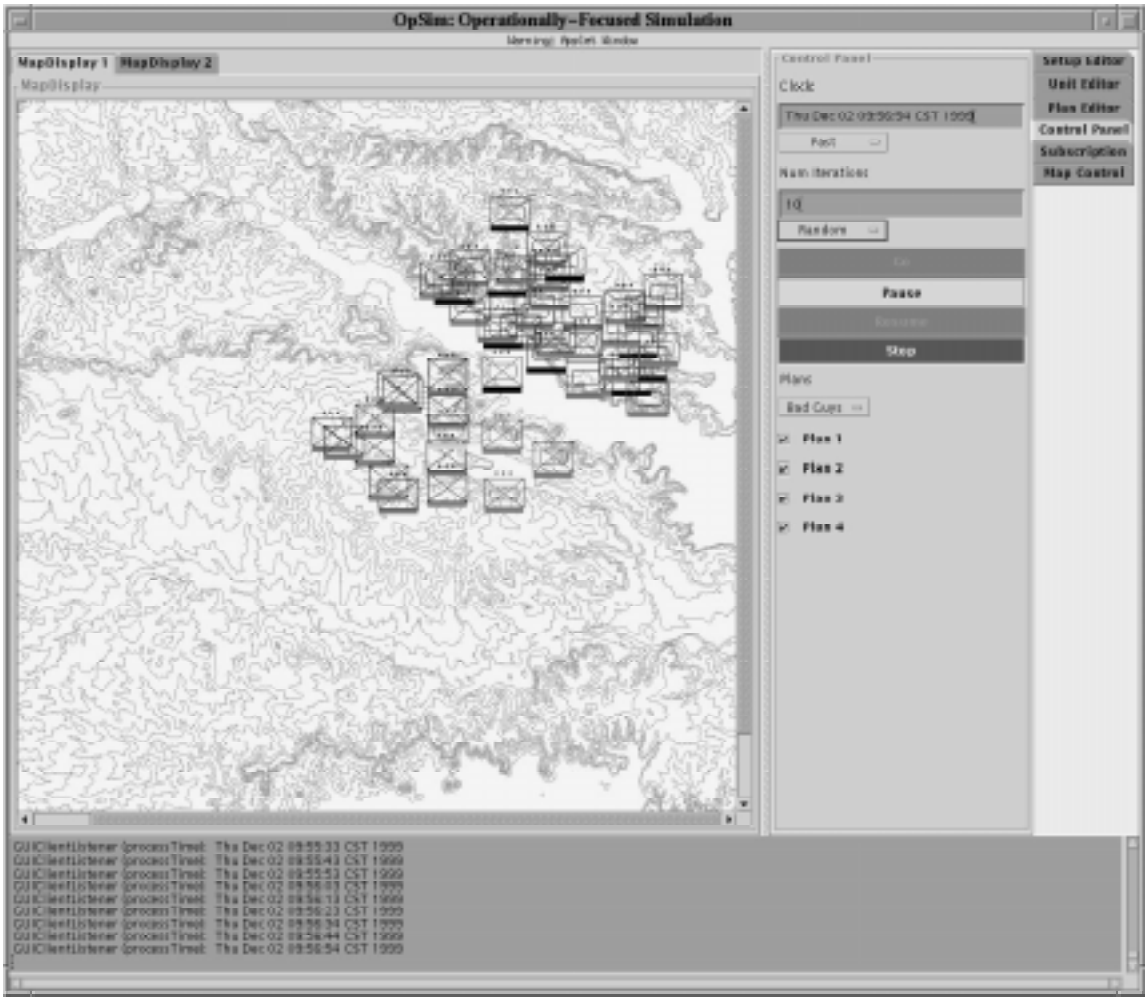


Figure 15: Partially completed attack by a friendly brigade against an enemy battalion

distributions for the various entities were changed. In the first experiment the family of distribution remained the same, but the parameters were changed. Over time, the Results Monitor detected the differences in performance and recommended that the parameters in the "simulated" OpSim be changed to ones closer to those in the "real"

OpSim. In the second experiment, the family of distribution was changed. As designed, the Results Monitor reported a recommended change of distribution family, and the Top Level Monitor created a dialog box to ask the user to confirm this change.

The second set of experiments was designed to ensure that there were no false reports of problems. Three scenarios were run in which the "simulated" OpSim and the "real" OpSim ran the same plans with the same parameters. The two simulations were both run in random mode, so that there were slight differences between the outcomes of the two simulations, but the differences were small. As expected, the Operations Monitors did not report any concerns or warnings.

The third set of experiments was designed to stimulate the Operations Monitors. This was done by adding additional enemy entities to the "real" OpSim that were not present in the "simulated" OpSim. At first a very large number of enemy entities was added, with the intention of creating a situation that clearly indicated the plan was at risk. The Forces Monitors very quickly detected the real operation diverging from the plan and launched simulations to explore the differences. The Simulation Monitors reported heavy losses, which were interpreted by the fuzzy rule bases as significant. Finally, the Top Level Monitor, collecting the reports from its children, opened a dialog box to report the warning to the commander.

The number of additional enemy entities in the "real" operation was slowly reduced to just two tank platoons. (This scenario is shown in Figures 15 and 16.) When the two currently serving Army officers looked at the scenario, they were unable to definitively classify the threat as significant or insignificant. As the battle progressed,

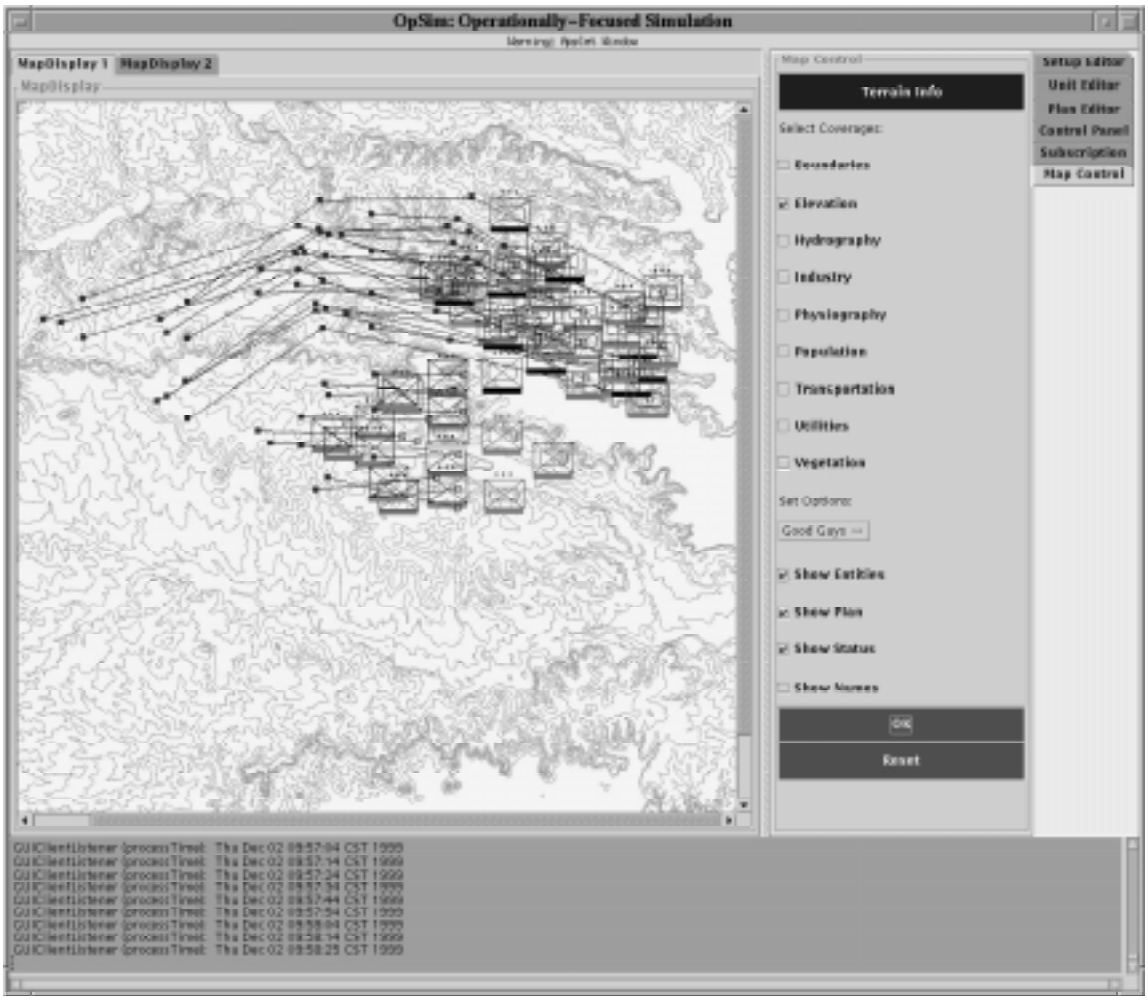


Figure 16: Partially completed attack with two extra enemy tank platoons and routes of march shown

the light infantry units in the south took heavier casualties in the real operation than in the plan. The Blue Forces Monitor launched a Simulation Monitor to explore the differences. The "real" OpSim was run in random mode, so the outcome of each battle was different. If the enemy was doing particularly well (good die rolls, so to speak) in a particular experiment, the monitors would inform the commander that there was a

concern. With just two additional tank platoons the predicted outcome did not warrant a warning, just a concern.

This last experiment is the most interesting, as it provides the best demonstration of the feasibility of the proposed methodology. The human experts were unsure of the impact of the additional two enemy platoons. Even after casualties were reported, and the friendly infantry units were taking losses, the significance of the losses was unclear. The immediate reaction on seeing two light infantry platoons jump from green to red status was that the plan was at risk. While the increase in casualties was alarming, it had only a moderate impact on the outcome of the operation. The friendly forces of course lost more personnel and equipment, but they reached their eventual objective in strength.

D. Verification

Verification is the process of ensuring that the system performs as designed. This is essentially an issue of software writing and testing. Does the code do what the designer wanted it to do? Verification of the implementation was done in stages as the code was being written.

1. OpSim

The purpose of this work was not to build an approved attrition model; however, OpSim required an attrition model to perform its functions. Two different attrition models were implemented, the Dupuy QJM model and an *ad hoc* model based on recreational war games. The implementation of OpSim is sufficiently flexible that other

attrition models could be used instead of either of the two currently implemented ones. Because the terrain effects have not been implemented in the prototype OpSim (see Chapter IV), it was difficult to test the attrition model. The basic rule of thumb that an attacker should have three to one odds against an enemy to attack was used to test the various combat results. Given a scenario in which the attacker had three-to-one odds against a heavily entrenched defender, one would expect the attacker to win approximately 50% of the time. This result was seen for both of the implemented attrition models.

2. Operations Monitors

An often-used verification technique is to compare the answers given by the new system with the known correct answer. This technique was used to test the various inference mechanisms used by the different Operations Monitors. As described in Chapter IV, MATLAB was used to create the fuzzy rule bases. The MATLAB Fuzzy Toolbox provides the capability for the user to specify input values to the fuzzy rule base and determine the computed output value. Assuming that the MATLAB implementation of the various fuzzy combination schemes (e.g., Mamdani and Segueno) is correct, these answers from MATLAB were treated as the correct answers. Then the fuzzy implementation of SAM developed for this research was stimulated with the same input values. The output values were the same (to three decimal places) as those provided by MATLAB. Since the implementation for this research used a different fuzzy combination method, SAM, than MATLAB, the discrepancy beyond the third decimal place was determined to be caused by the difference in implementations and floating

point round-off error. (The implementation of SAM used in this research used all double-precision floating point math.)

Testing the crisp rule base was more difficult. Fortunately the crisp rule bases were small. Several test cases were built. The facts that corresponded to these test cases were asserted into the rule base. The output of the crisp rule base was compared to the known correct answer for the test case. Recall that the crisp rule base was used merely to control the actions of the monitors, not to classify differences between the real operation and the plan. One test case involved asserting the results from a simulation experiment into a Forces Monitor's rule base to see if it would recommend report the results to its parent. During the experiments described earlier, the various monitors performed as designed.

E. Validation

Validation is the process of ensuring that the design of the system solves the desired problem. Unlike verification, validation is used to determine whether the designed system is useful to the target consumer. Validity is also used to decide whether the system provides answers that "make sense" or are useful to any available human experts. Validation was done anecdotally, based on input from currently-serving Army officers. Due to lack of systems with which to compare the proposed methodology and the lack of practical availability of domain experts to evaluate the system, only face validity was shown [10, 11].

1. OpSim

In Chapter II the motivation behind this research was explained. Training simulations have been used to facilitate course of action development and analysis in isolated incidents and experiments [1, 3, 9, 22, 36, 40, 58, 60]. The worth of a "scientific" method of creating and assessing the value of courses of action during planning is well recognized [1, 3, 9, 22, 40, 58, 60]; however, no simulation focused solely on this task has been developed.

There is wide recognition within the Army and Department of Defense that a system to facilitate rapid course of action analysis is needed. The design of the overall methodology and the requirements for the operationally focused simulation support the use of OpSim in this role. In addition, because OpSim supports the querying of information and subscribing to information, it makes possible the use of software agents to help monitor the operation. The technology used to pass this information is less important than the existence of this capability. OpSim uses message passing; however, CORBA, Persistent Object Protocol [39], Java RMI, or some future technology could be substituted. The use of ASCII-based message across TCP/IP sockets was designed to allow the easy creation of Operations Monitors without specifying an implementation language.

OpSim provides feedback to a planner. OpSim estimates expected losses when executing various friendly courses of action versus various enemy courses of action. This estimation can be done before an operation or during the operation. It also provides

some feedback about how long it will take to accomplish the mission. These are pieces of information often desired by planners [62, 63].

Given an attrition model that the users can accept, OpSim provides answers that "make sense." The answers OpSim provides are necessary, if not sufficient, for planners to decide between courses of action. This information is also of interest to commanders and staffs. The design of OpSim and its implementation are, therefore, valid.

2. Operations Monitors

The validity of the overall methodology and the Operations Monitors was demonstrated by the prototype system. Given an ambiguous situation, the collection of operations monitors provided information to the commander that would be useful in a real operation. As the Operations Monitors discover significant differences between the real operation and the planned operation, they use other tools (simulation in the prototype) to predict the effect of these differences on the outcome of the operation. This would be of significant benefit to commanders. The active-duty Army officers who were asked to evaluate the prototype recognized this benefit.

F. Future Work

The prototype OpSim and Operations Monitors were never intended to be used in the field. They were intended to demonstrate the feasibility of the overall methodology. The creation of a fieldable combat simulation is a difficult task, often requiring teams of programmers several years. Similarly, the inference mechanisms associated with each of

the various Operations Monitors will probably become quite complex by the time a fieldable system is developed. Each Operations Monitor might require its own research and development effort. This research has demonstrated the usefulness of a system with the capabilities described (and implemented to prototype levels). A large amount of work is needed before such a system could be given to war fighters.

1. Development of a More Comprehensive Combat Simulation

The prototype OpSim provides some sophisticated and useful information to commanders and planners. Before OpSim would be ready for field use, many more capabilities are needed. It is unclear whether it would be easier to trim down an existing Army simulation to just those components needed for an operationally focused simulation and then add needed capabilities or whether it would be easier to build a new operationally focused simulation. Assuming that one would proceed by enhancing OpSim to a fieldable state, below is a short list of needed enhancements.

a) Terrain

The use of off-the-shelf terrain is important for a tool to be used in the field. In high-tempo operations, there will never be time for the custom construction of a terrain database. Whether NIMA will continue to work toward universal acceptance of the VPF terrain products or whether some other solution will become the terrain of choice is unclear. The prototype implementation of OpSim reads VMAP-2 terrain databases and displays them. In order to make OpSim more useful, it must have the capability to execute spatial queries of the terrain database.

b) Coalitions

United States military forces are increasingly deployed as part of coalition operations. The prototype OpSim has the capability to represent multiple forces; however, any force not one's own is considered to be an enemy. OpSim should be enhanced so that forces can have relationships other than enemy, such as neutral, friendly, etc. These relationships should be represented in such a way that the relationship need not be symmetrical. For instance, side A might think that side B is neutral, but side B might consider side A to be an enemy. This enhancement would make OpSim appropriate for a wide spectrum of military operations, not just two-sided conventional fights.

c) Operations Other Than Direct Fire

Recall that the prototype OpSim only simulates direct-fire ground engagements. Modern military operations include aircraft, anti-aircraft weapons, indirect fire weapons (e.g., artillery and mortars), precision-guided munitions (including sensor-to-shooter links), mine fields, chemical weapons, and obstacles. These aspects of modern warfare must be represented in OpSim in order to make it useful in the field. In addition, tactical communications networks, both voice and data, must be represented, because the loss or damage of these networks has a significant impact on the outcome of operations.

d) Logistics

German Field Marshal Irwin Rommel once said that amateurs study tactics and professionals study logistics. Often the outcome of an operation depends as much on whether fuel, ammunition, and food can be delivered to the soldiers as it does on the results of individual combats. Logistics should be represented within the simulation to provide better information to the commander and staff. During planning, the representation of logistics would help the logistics officers determine the number and location of re-supply points, supply routes, etc. During the execution of the operation, a logistics Operations Monitor could monitor the number of tank rounds, for instance, remaining in a platoon versus the planned number. If the tank unit appeared to be using ammunition at a faster-than-planned rate, a Simulation Monitor could be used to determine if that unit will run out of ammunition prior to the end of the operation.

e) Better Interface

The user interface is adequate for a prototype. While some care was taken in the design of the GUI to make it easy to input plans, clone plans, run the simulation, and display the resulting statistics, it could be much better. No right-click functionality has been implemented, so it takes three mouse clicks to edit an entity's capabilities. Entity icons cannot be dragged around the map; this also takes several mouse clicks. When the prototype was begun, the Java Swing API had not been released. Many of the Java Swing functions were added to the interface late in development, but a redesign is needed. The design of the GUI was not the focus of this research, so it received little

attention compared to the design of the simulation and monitors. While the current GUI is not cumbersome, it is essential that the best practices in user interface design be used to ensure that commanders and staffs can concentrate on the plan and not the tool.

f) Better Statistical Computations

The statistics collector computes all the means and confidence intervals after the various simulation experiments are completed. Statistics are computed for each pair of sides in the scenario. In other words, if there are three sides in the scenario, A, B, and C, A-B, A-C, and B-C comparisons would be computed. Any of these pairwise comparisons can be shown in a display like the one in Figure 12; however, only the side A versus side B comparison is displayed in the prototype. To be more useful to planners, the planners should be able to tailor the statistics that are computed and displayed. In addition, it would be useful if statistics were computed on an ongoing basis rather than merely at the end. This capability would be most useful when monitoring the current operation.

2. Using a Database Management System

The implementation of the entity and node lists was discussed in Chapter IV. While the use of a hash table to hold these lists allows fast access to the contents, a better long-term solution would be the use of a database management system. With an SQL-compliant database, the developers of the simulation and the monitors would not need to anticipate in advance all possible types of information that OMs might need. When an OM needs a new piece of information -- if the data is maintained within the simulation –

the OM developer would merely need to create a new database query. Using a database management system as the underlying state representation would make the overall system more flexible.

3. Be a Fixer, Not Just a Finder

One of the stated objectives of the proposed methodology is to decrease the amount of information that planners and commanders have to process. While the task of identifying when the plan is at risk is not trivial, a much more difficult problem is for the system to then recommend a solution. This relates to the course of action generation research conducted by Fiebig, Hayes, and others [31, 32, 130]. Solving this problem requires the creation of a detailed and complete language for describing courses of action as well as the development of techniques for generating them. Courses of action are represented as plans in OpSim. It is unclear whether this representation would be useful in a tool designed to invent new courses of action for an operation. This course of action generation capability would be useful during planning. It would also be very useful during the conduct of the operation. After the Top Level Monitor identified that the plan was at risk, it could present the commander with some number of recommended alternative solutions.

4. Complete the Mapping of Mission Tasks to Operations Monitors

The principle that determines which OMs are automatically launched requires a mapping from mission tasks to a default set of OMs. This was only done for two tasks, seize and defend in sector, during the construction of the prototype system. The Army's

Maneuver Control System (part of the ABCS [9]) contains an operations order generation tool. This tool lists all the mission tasks. Each of these tasks should be analyzed to create a default set of OMs for each task.

5. Develop More Operations Monitors and Tools

Only a small number of OMs was developed as part of this prototype. The first step would be to flesh out the OMs that are merely stubs, such as Fire Support, Logistics, Engineering, and Air Defense in the first level of the hierarchy. (This clearly requires that these Battlefield Functions be represented in OpSim.) In the prototype hierarchy, the time and mission monitors under the Maneuver Monitor were not implemented. In most cases what information OpSim was incapable of providing determined which OMs were not implemented.

Each OM could be the subject of its own research project to determine all the information needed to make inferences about the plan versus the real operation. As part of this research, the appropriate reasoning mechanism should be identified. For example, as stated in Chapter IV, the Top Level Monitor adds all evidence together to determine whether the operation is likely to succeed. There may be instances in which the combination of two pieces of evidence is greater than the sums of the parts. The OMs must be robust enough to deal effectively with uncertainty while making quick, accurate assessments of the situation.

CHAPTER VI

SUMMARY AND CONCLUSIONS

A. Summary

The purpose of this research was to develop a methodology for the use of simulation during combat operations. This methodology involves the use of an operationally focused simulation and software agents to perform analysis and make recommendations.

The operationally focused simulation was designed to facilitate course of action analysis and help select the best course of action. An operationally focused simulation could also be used to help orchestrate rehearsals. Once the operation has begun, the simulation is run in near real time, paralleling the current operation. The software agents, called Operations Monitors (OMs), periodically compare events in the current operation with those in the simulation of the chosen course of action. When significant differences exist, the OMs can launch subordinate OMs and other tools to perform analysis. If the OMs determine that something in the current operation puts the final success of the mission at risk, the OMs advise the commander. The OMs are also used to update the parameters of the simulation to make it a better predictor of combat results in the future. After the operation ends, the operationally focused simulation can be used to help conduct after-action analysis.

The OMs exist in a hierarchy, and each OM focuses on a narrow portion of the problem domain. The first level in the hierarchy is based on the battlefield functions described in Army Field Manual 100-5 [142]. The second level in the hierarchy is based on the mnemonic METTTC used by Army planning staffs. The OMs analyze the differences between the current operation and the plan. Then they report a summary of their findings to their parent OM. The top-level OM decides whether to inform the commander of problems. Reports to the commander are either concerns or warnings. Concerns indicate that there are moderate differences between the planned outcome and the expected outcome of the operation, but the mission will still be a success. Warnings indicate that there are significant differences between the planned outcome and the expected outcome, and the accomplishment of the mission is at risk.

The capability exists in current command and control systems to provide more information to the commander than anyone can process. The purpose of a system like the one proposed in this research is to focus the commander's attention on those aspects of the operation that have significant impact on the success of the mission. Given such a system commanders and staffs will be able to make better decisions faster than the enemy does.

The implemented prototype shows that the overall methodology is sound. During planning, the operationally focused simulation, OpSim, provides useful feedback about average losses expected for each friendly course of action against each predicted enemy course of action. This provides input to the planning process for choosing one course of action from several. Once the operation begins, OpSim runs the planned

operation in near real time while the real operation is being conducted. The various OMs compare the planned operation with the real operation, focusing on their narrow domain, such as enemy forces. When needed, these OMs launch subordinate OMs to project the current operation to end state and determine if the probability of success is at risk. The top-level monitor computes a utility function based on reports from subordinate OMs and decides whether to inform the commander. The testing of this system indicated that the system could more accurately predict the effects of differences between the plan and the actual operation than could human domain experts.

B. Significance of Research

The principle contribution of this research is to provide a methodology for the use of simulation during operations. This research addresses the following issues:

- The requirements (from a tactical, simulation, and software agent perspective) of such a system. One of the reasons that no such system has previously been built is that the requirements have never before been articulated. Having developed the requirements, many of which have been corroborated by other research since the requirements were first proposed, this research makes possible the construction of other systems for monitoring current operations.
- Dynamic adaptation of the system in general (and the simulation in particular) over time. With the feedback and adaptation mechanism described in this system and implemented in the prototype, the simulation

eventually becomes a better tool for predicting the outcome of courses of action.

- Dynamic invocation of software agents to conduct analysis. This methodology proposed a principle that guides the minimal set of Operations Monitors that are invoked when the system is first launched. A mapping between missions and default sets of OMs is used. OMs can then launch other agents as necessary to conduct further analysis.
- Decentralized Analysis of aggregate information. The job of the highest-level monitor is made simpler by the fact that lower-level monitors conduct filtering and analysis at their levels. Rather than sorting through all the available information to make decisions, the highest-level monitor makes decisions based on reports from subordinates. This makes the analysis of a very complex problem tractable.

The overall benefit of this research is to build a foundation on which other tools and software agents can be built. This methodology will support quick, accurate assessment of the rapidly changing operational environment. The implemented prototype demonstrates the feasibility of this approach. Given a production system with the capabilities of the prototype system, a commander and staff could use available information significantly better than an enemy could.

REFERENCES

[1] P. J. Delany, Personal Communication (regarding Army Modeling and Simulation Office Policy and Technology Working Group video teleconference), Captain, U.S. Army, Army Modeling and Simulation Office, 28 October 1998.

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

[3] D. Gunning, "Command Post of the Future," Available at http://www.mole.dc.isx.com/cpof, 15 October 1998.

[4] C. v. Clausewitz, *On War,* Princeton, NJ: Princeton University Press, 1984.

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

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

[7] 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.

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

[9] J. E. Bessler, "Army Battle Command System (ABCS) Capstone Requirements Document (CRD)," Tech. Rep. Revision 1.0, TPIO-ABCS, Ft. Leavenworth, KS, 10 February 1998.

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

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

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

[13] B. P. Zeigler, "Review of Theory in Model Abstraction," *Proceedings of SPIE*, vol. 3369, no. 1, 1998, pp. 2-12.

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

[15] G. Booch, *Object-Oriented Analysis and Design*, Menlo Park, CA: Addison-Wesley Publishing Company, 1994.

[16] D. Caughlin and A. F. Sisti, "A Summary of Model Abstraction Techniques," *Proceedings of SPIE*, vol. 3083, no. 1, 1997, pp. 2-12.

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

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

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

[20] J. Banks, "Principles of Simulation," in *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*, J. Banks, ed., New York: John Wiley & Sons, Inc., 1998, pp. 1 - 30.

[21] 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.

[22] 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.

[23] W. J. Davis, "Developing an Intelligent Controller for Integrated Manufacturing Resource Planning," Tech. Rep. Draft, University of Illinois at Urbana-Champaign, Urbana, IL, October 1998.

[24] W. J. Davis, "A Framework for the Distributed Intelligent Control of Advanced Manufacturing Systems," Tech. Rep. Draft, University of Illinois at Urbana-Champaign, Urban, IL, October 1998.

[25] 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, January 1998, pp. 40 - 53.

[26] M. Andersson and G. Olsson, “A Simulation Based Decision Support Approach for Operational Capacity Planning in a Customer Order Driven Assembly Line,” in *Proc. Winter Simulation Conference*, Washington, DC, 13-16 December 1998, pp. 935-941.

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

[28] S. Andradottir, “Simulation Optimization,” in *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*, J. Banks, ed., New York: John Wiley & Sons, 1998, pp. 307-333.

[29] J. R. Surdu and U. W. Pooch, “Connecting the Operational Environment to Simulation,” in *Proc. Advanced Simulation Technology Conference: Military, Government, and Aerospace Simulation*, San Diego, CA, 11-14 April 1999, pp. 94-99.

[30] 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 *Military, Government and Aerospace 1999*, San Diego, CA, April 11-15 1999, pp. 3-8.

[31] C. B. Fiebig and C. C. Hayes, “DAISY: A Design Methodology for Experience-Centered Planning Support Systems,” in *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, San Diego, CA, 11-14 October 1998, pp. 920-925.

[32] C. C. Fiebig, C. C. Hayes, and J. Schlabach, “Human-Computer Interaction Issues in a Battlefield Reasoning System,” in *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, Orlando, FL, 12-15 October 1997, pp. 3204-3209.

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

[34] P. Perla, *The Art of Wargaming*, Annapolis, MD: Naval Institute Press, 1990.

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

[36] C. L. Blais and W. M. Garrabrants, “Simulation in Support of Mission Planning,” in *Proc. Advanced Simulation Technology Conference: Military, Government, and Aerospace*, San Diego, CA, 11-17 April 1999, pp. 117-122.

[37] J. M. Brennan, “The Army Experiment 4 (AE4) Simulation to Army Battle Command System (ABCS) Experiments,” Tech. Rep. 98-9-1, STRICOM, Orlando, FL, September 1998.

[38] B. Brown, “RTM Integration Phase I Test Plan,” Tech. Rep. Coordinating Draft, National Simulation Center, Ft. Leavenworth, KS, February 1998.

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

[40] P. A. Fishwick, G. Kim, and J. J. Lee, “Improved Decision Making Through Simulation Based Planning,” Tech. Rep. 94-14, Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL, February 1994.

[41] J. J. Lee and P. A. Fishwick, “Simulation-Based Planning for Computer Generated Forces,” Tech. Rep. 96-14, Department of Computer and Information Sciences and Engineering, University of Florida, Gainesville, FL, February 1996.

[42] R. L. Helmbold, “Rates of Advance in Land Combat Operations,” in *Warfare Modeling*, J. Braken, M. Kress, and R. E. Rosenthal, ed., New York: John Wiley & Sons, 1995, pp. 493-528.

[43] J. R. Scales, “A Modified Lanchester Linear Process Calibrated to Historical Data,” in *Warfare Modeling*, J. Bracken, M. Kress, and R. E. Rosenthal, ed., New York: John Wiley & Sons, 1995, pp. 345-356.

[44] J. Bracken, “Lanchester Models of the Ardennes Campaign,” in *Warfare Modeling*, J. Bracken, M. Kress, and R. E. Rosenthal, ed., New York: John Wiley & Sons, 1995, pp. 417-427.

[45] J. Bracken, M. Kress, and R. E. Rosenthal, “Introduction,” in *Warfare Modeling*, J. Bracken, M. Kress, and R. E. Rosenthal, ed., New York: John Wiley & Sons, Inc., 1995, pp. 1 - 4.

[46] D. S. Hartley and R. L. Helmbold, “Validating Lanchester's Square Law and Other Attrition Models,” in *Warfare Modeling*, J. Bracken, M. Kress, and R. E. Rosenthal, ed., New York: John Wiley & Sons, 1995, pp. 467-492.

[47] D. S. Hartley, "A Mathematical Model of Attrition Data," in *Warfare Modeling*, J. Bracken, M. Kress, and R. E. Rosenthal, ed., New York: John Wiley & Sons, 1995, pp. 443-466.

[48] J. Yang and A. V. Gafarian, "A Fast Approximation of Homogeneous Stochastic Combat," in *Warfare Modeling*, J. Bracken, M. Kress, and R. E. Rosenthal, ed., New York: John Wiley & Sons, 1995, pp. 357-392.

[49] I. David, "Lanchester Modeling and the Biblical Account of the Battle of Gibeah," in *Warfare Modeling*, J. Bracken, M. Kress, and R. E. Rosenthal, ed., New York: John Wiley & Sons, 1995, pp. 437-442.

[50] R. D. Fricker, "Attrition Models of the Ardennes Campaign," *Naval Research Logistics*, vol. 45, no. 1, January 1997, pp. 1-22.

[51] T. N. Dupuy, *Attrition: Forecasting Battle Casualties and Equipment Losses in Modern War*, Falls Church, VA: Nova Publications, 1995.

[52] T. N. Dupuy, *Numbers, Predictions, and War: Using History to Evaluate Combat Factors and Predict the Outcome of Battles*, Indianapolis, IN: The Bobbs-Merrill Company, Inc., 1979.

[53] D. Denning, *Information Warfare and Security*, Reading, MA: Addison-Wesley, 1999.

[54] C. A. Carver, "Information Warfare: Task Force XXI or Task Force Smith?," *Military Review*, vol. 98, no. 4, September - November 1998, pp. 26-30.

[55] S. Nitzberg, "The Cyber Battlefield - Is This the Setting for the Ultimate World War?," in *Proc. International Symposium Technology and Society*, Glasgow, Scotland, June 9-13 1997, pp. 1-7.

[56] U. S. Army, *FM 100-6: Information Operations*, Washington, DC: Headquarters, Department of the Army, 1996.

[57] A. Sayles, J. Marin, and D. Welch, Personal Communication (regarding A Briefing on the Proposed Information Operations Technology Center), Department of Electrical Engineering and Computer Science, United States Military Academy, West Point, NY, 13 May 1999.

[58] R. L. Bateman, "Avoiding Information Overload," *Military Review*, vol. 98, no. 3, July - August 1998, pp. 53-58.

[59] J. W. McLendon, "Information Warfare: Impacts and Concerns," in *Battlefield of the Future: 21st Century Warfare Issues*, B. R. Schneider and L. E. Grinter, ed., Maxwell Air Force Base, AL: Air War College, 1995, pp. 171-199.

[60] R. J. Bunker, "Information Operations and the Conduct of Land Warfare," *Military Review*, vol. 98, no. 4, September - November 1998, pp. 4-17.

[61] J. McKitrick, J. Blackwell, F. Littlepage, G. Kraus, R. Blanchfield, and D. Hill, "The Revolution in Military Affairs," in *Battlefield of the Future: 21st Century Warfare Issues*, B. R. Schneider and L. E. Grinter, ed., Maxwell Air Force Base, AL: Air War College, 1995, pp. 65-101.

[62] U. S. Army, *ST 100-9: Techniques and Procedures for Tactical Decisionmaking*, Fort Leavenworth, KS: U.S. Army Command and General Staff College, 1991.

[63] U. S. Army, *FM 101-5: Staff Organization and Operations*, Washington, DC: Headquarters, Department of the Army, 1998.

[64] P. Wyden, *Bay of Pigs*, New York: Simon and Schuster, 1979.

[65] 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 Third International Workshop on Agent Theories, Architectures, and Languages*, J. P. Müller, M. J. Wooldridge, and N. R. Jennings, ed., Berlin: Springer-Verlag, 1997, pp. 21-35.

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

[67] R. A. Brooks, "A Robust Layered Control System for a Mobile Robot," Tech. Rep. AIM-864, Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA, September 1985.

[68] R. A. Brooks, "Elephants Don't Play Chess," in *Designing Autonomous Agents*, P. Maes, ed., Cambridge, MA: MIT Press, 1990, pp. 3-18.

[69] R. A. Brooks, "Intelligence Without Reason," in *Proc. 12th Intl. Joint Conference on Artificial Intelligence*, R. M. a. J. Reiter, ed., Sydney, Australia: Morgan Kaufmann, 1991, pp. 569-595.

[70] O. Etzioni, "Intelligence Without Robots (A Reply to Brooks)," *AI Magazine*, vol. 14, no. 4, April 1993, pp. 7-13.

[71] M. H. Coen, "SodaBot: A Software Agent Environment and Construction System," Tech. Rep. AI 1493, Massachusetts Institute of Technology, Cambridge, MA, June 1994.

[72] M. H. Coen, "Building Brains for Rooms: Designing Distributed Software Agents," in *Ninth Conference on Innovative Applications of AI (IAAI97)*, Providence, RI, 27-31 July 1997, pp. 1-7.

[73] W. Becket and N. I. Badler, "Integrated Behavioral Agent Architecture," in *Proc. 3rd Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL, March 1993, pp. 57-68.

[74] P. Maes, "How to Do the Right Thing," Tech. Rep. 1180, Massachusetts Institute of Technology, Cambridge, MA, January 1989.

[75] P. Maes, "Modeling Adaptive Autonomous Agents," *Artificial Life*, vol. 1, no. 1, Fall 1994, pp. 135-162.

[76] P. Maes, "Agents That Reduce Work and Information Overload," *Communications of the ACM*, vol. 37, no. 7, July 1994, pp. 31-41.

[77] P. Maes, "Situated Agents Can Have Goals," in *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, P. Maes, ed., Cambridge, MA: MIT Press, 1994, pp. 49-70.

[78] H. Nwana, "Software Agents: An Overview," *Knowledge Engineering Review*, vol. 11, no. 3, November 1996, pp. 205-244.

[79] M. Williamson, K. Decker, and K. Sycara, "Executing Decision-theoretic Plans in Multi-agent Environments (Extended Abstract)," Available at http://www.cs.cmu.edu/~softagents/publications.html, 27 April 1999.

[80] T. Grimm, J. Mitloehner, and W. Schoenfeldinger, "Bounded Rationality and Adaptive Agents in Economic Modeling," in *Proc. International Conference on Applied Programming Languages*, San Antonio, TX, June 1995, pp. 56-62.

[81] T. W. Sandholm and V. R. Lesser, "Utility-Based Termination of Anytime Algorithms," Tech. Rep. CMPSCI 94-54, University of Massachusetts at Amherst, Amherst, MA, July 1994.

[82] A. Doan and P. Haddawy, "Generating Macro Operators for Decision-Theoretic Planning," in *Extending Theories of Action: Formal Theory & Practical Applications: Papers from the 1995 AAAI Spring Symposium*, ed., Menlo Park, CA: AAAI Press, 1995, pp. 68-73.

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

[84] S. J. Henkind and M. C. Harrison, "Analysis of Four Uncertainty Calculi," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, no. 5, May 1989, pp. 700-714.

[85] T. M. Mitchell, *Machine Learning*, Boston, MA: McGraw-Hill, 1997.

[86] J. Yen and R. Lengari, *Fuzzy Logic: Intelligence, Control and Information*, New York: Prentice Hall, 1999.

[87] D. J. Ragsdale, C. D. Butler, B. A. Cox, J. Yen, and U. W. Pooch, "Fuzzy Logic Approach for Intelligence Analysis of Actual and Simulated Military Reconnaissance Missions," in *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, Orlando, FL, 12-15 October 1997, pp. 60-65.

[88] J. Dayhoff, *Neural Network Architectures*, New York: Van Nostrand Reinhold, 1990.

[89] V. W. Porto, L. J. Fogel, and D. B. Fogel, "Evolving Tactics in Computer-Generated Forces," in *Proc. Enabling Technologies for Simulation Science III*, Orlando, FL, 7-11 April 1999, pp. 75-80.

[90] F. Ygge and H. Akkermans, "Making a Case for Multi-Agent Systems," in *Proc. Eights Modeling Autonomous Agents in a Multi-Agent World*, Ronneby, Sweden, 13-16 May 1997, pp. 156-176.

[91] M. R. Genesereth and S. P. Ketchpel, "Software Agents," *Communications of the ACM*, vol. 37, no. 7, July 1994, pp. 48-53, 147.

[92] J. M. Bradshaw, "Kaos: Toward an Industry-Strength Open Agents Architecture," in *Software Agents*, J. M. Bradshaw, ed., Menlo Park, CA: AAAI Press, 1997, pp. 375-418.

[93] T. Finin, R. Fritzson, D. McKay, and R. McIntire, "KQML as an Agent Communication Language," in *Proc. Third International Conference on Information and Knowledge Management*, Washington, DC, 29 November - 2 December 1992, pp. 456-463.

[94] K. S. Decker and V. R. Lesser, "Generalizing the Partial Global Planning Algorithm," *International Journal of Intelligent Cooperative Systems*, vol. 1, no. 2, February 1992, pp. 319-346.

[95] L. Ekenberg, M. Boman, and M. Danielson, “Tool for Coordinating Autonomous Agents with Conflicting Goals,” in *Proc. First International Conference on Multi-Agent Systems*, San Francisco, CA, 12-14 June 1995, pp. 89-93.

[96] K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng, “Distributed Intelligent Agents,” *IEEE Expert*, vol. 11, no. 6, June 1999, pp. 36-46.

[97] J. Anderson and M. Evans, “Supporting Flexible Autonomy in a Simulation Environment for Intelligent Agent Designs,” in *Proc. Fourth Annual Conference on AI, Simulation, and Planning in High Autonomy Systems*, Tucson, AZ, 20-22 September 1993, pp. 60-66.

[98] V. R. Lesser, “Reflections on the Nature of Multi-Agent Coordination and Its Implications for an Agent Architecture,” Tech. Rep. CMPSCI 98-10, University of Massachusetts at Amherst, Amherst, MA, February 1998.

[99] Q. Yang, *Intelligent Planning: A Decomposition and Abstraction Based Approach*, Berlin: Springer-Verlag, 1997.

[100] E. Rich and K. Knight, *Artificial Intelligence*, New York: McGraw-Hill, 1991.

[101] E. D. Sacerdoti, “The Nonlinear Nature of Plans,” in *Proc. International Joint Conferences on Artificial Intelligence*, Tibilisi, Georgia, USSR, 23-25 August 1975, pp. 206-214.

[102] P. E. Agre and D. Chapman, “What Are Plans For?,” in *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, P. Maes, ed., Cambridge, MA: MIT Press, 1991, pp. 17-34.

[103] L. P. Kaelbling and S. J. Rosenschein, “Action and Planning in Embedded Agents,” in *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, P. Maes, ed., Cambridge, MA: MIT Press, 1991, pp. 35-48.

[104] C. Boutilier, T. Dean, and S. Hanks, “Planning Under Uncertainty: Structural Assumptions and Computational Leverage,” in *New Directions in AI Planning*, M. Ghallab and A. Milani, ed., Amsterdam, The Netherlands: IOS Press, 1996, pp. 157-171.

[105] B. Drabble and A. Tate, “O-Plan: A Situated Planning Agent,” in *New Directions in AI Planning*, M. Ghallab and A. Milani, ed., Amsterdam, The Netherlands: IOS Press, 1996, pp. 247-259.

[106] E. Fink and M. Veloso, “Formalizing the PRODIGY Planning Algorithm,” in *New Directions in AI Planning*, M. Ghallab and A. Milani, ed., Amsterdam, The Netherlands: IOS Press, 1996, pp. 261-271.

[107] M. Veloso, J. Carbonell, A. Perez, D. Borrajo, E. Fink, and J. Blythe, “Integrating Planning and Learning: The PRODIGY Architecture,” *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 7, no. 1, July 1995, pp. 81-120.

[108] 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, The Netherlands: IOS Press, 1995, pp. 103-112.

[109] A. L. Lansky, “Action-Based Planning,” in *Proc. Second International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, University of Chicago, Chicago, IL, June 1994, pp. 110-115.

[110] C. A. Ntuen, E. H. Park, M. E. McBribe, W. Marshak, D. N. Mountjoy, and P. L. Yarbrough, “Collective Asset Display for Commander's Decision Making,” in *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, San Diego, CA, 11-14 October 1998, pp. 1302-1305.

[111] M. G. Oxenham, D. J. Kewley, and M. J. Nelson, “Performance Assessment of Data Fusion Systems,” in *Proc. First Australian Data Fusion Symposium*, Adelaide, South Australia, 21-21 November 1996, pp. 36-41.

[112] S. C. A. Thomopoulos, T. W. Hilands, and B. H. Chen, “Applications of Joint Detection/Estimation Filter to Data Fusion,” in *Proc. 1993 International Conference on Systems, Man, and Cybernetics*, Le Touquet, France, 17-20 October 1993, pp. 678-683.

[113] J. Llinas and D. L. Hall, “An Introduction to Multi-Sensor Data Fusion,” in *Proc. 1998 IEEE International Symposium on Circuits and Systems*, Monterrey, CA, 31 May - 3 June 1998, pp. 537-540.

[114] J. K. Rosenblatt and C. E. Thorpe, “Combining Multiple Goals in a Behavior-Based Architecture,” in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, Pittsburgh, PA, 5-9 August 1995, pp. 136-141.

[115] R. R. Murphy, “Action-Oriented Sensor fusion for Telesystems,” in *Proc. National Telesystems Conference*, Washington, DC, 19-20 May 1992, pp. 10/11 - 10/14.

[116] I. Mayk, J. Salton, E. Dawidowicz, R. Wong, and L. Tran, “Distributed Situation Awareness for C2 Platforms,” in *Proc. IEEE International Conference on*

Systems, Man, and Cybernetics, Orlando, FL, 12-15 October 1997, pp. 4354-4359.

[117] R. E. Gibson, D. L. Hall, and J. A. Stover, "An Autonomous Fuzzy Logic Architecture for Multisensor Data Fusion," in *Proc. 1994 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, Las Vegas, NV, 2-5 October 1994, pp. 143-150.

[118] D. Kuncicky, S. Hruska, and R. C. Lacher, "Hybrid Systems: The Equivalence of Rule-Based Expert System and Artificial Neural Network Inference," *International Journal of Expert Systems*, vol. 4, no. 3, July - September 1992, pp. 281-297.

[119] R. C. Lacher, "Expert Networks: Paradigmatic Conflict, Technological Rapprochment," *Mind and Machines*, vol. 3, no. 1, January 1993, pp. 53-71.

[120] R. C. Lacher, S. Hruska, and D. Kuncicky, "Backpropagation Learning in Expert Networks," Tech. Rep. TR91-015, Department of Computer Science, Florida State University, Tallahassee, FL, March 1991.

[121] R. C. Lacher, S. I. Hruska, and D. C. Kuncicky, "Back-Propagation Learning in Expert Networks," *IEEE Transactions on Neural Networks*, vol. 3, no. 1, January 1992, pp. 62-72.

[122] O. Etzioni and D. Weld, "A Softbot-Based Interface to the Internet," *Communications of the ACM*, vol. 37, no. 7, July 1994, pp. 72-76.

[123] T. Mitchell, R. Caruana, D. Freitag, J. McDermott, and D. Zabowski, "Experience with a Learning Personal Assistant," *Communications of the ACM*, vol. 37, no. 7, July 1994, pp. 81-91.

[124] M. A. Hersh, "Sustainable Decision Making: The Role of Decision Support Systems," in *Proc. IEE Colloquium on Decision Making and Problem Solving*, London, England, 16 December 1997, pp. 611-615.

[125] F. S. Wong, P. Z. Wang, and T. H. Goh, "Fuzzy Neural Systems for Decision Making," in *Proc. IEEE 1991 International Joint Conference on Neural Networks*, Singapore, 18-21 November 1991, pp. 1625-1637.

[126] V. L. Sauter, "Intuitive Decision Making: Combining Advanced Analytical Tools with Human Intuition Increases Insight into Problems," *Communications of the ACM*, vol. 42, no. 6, June 1999, pp. 109-115.

[127] S. Zilberstein and V. Lesser, "Intelligent Information Gathering Using Decision Models," Tech. Rep. CMPSCI 96-35, University of Massachusetts at Amherst, Amherst, MA, March 1996.

[128] D. Rus, R. Gray, and D. Kotz, "Autonomous and Adaptive Agents That Gather Information," in *Proc. AAAI 96 Workshop on Intelligent Adaptive Agents*, Portland, OR, 4-8 August 1996, pp. 107-116.

[129] K. Decker, A. Pannu, K. Sycara, and M. Williamson, "Designing Behaviors for Information Agents," in *Proc. First International Conference on Autonomous Agents (AGENTS-97)*, Marina del Rey, CA, 5-8 February 1996, pp. 402-412.

[130] C. C. Hayes, H. Ergan, N. Tu, M. Liang, P. Jones, R. Barger, and D. Wilkins, "CoRAVEN: Intelligent Tools to Provide Multi-Perspective Decision Support and Data Visualization," Tech. Rep. UMN-IE-99-001, University of Minnesota, Minneapolis, MN, 14 December 1999.

[131] M. Liang and C. C. Hayes, "MIMOSA and Concept-Linker: Map Tools for Facilitating Intelligence Analysis," Tech. Rep. UMN-IE-99-002, University of Minnesota, Minneapolis, MN, May 1999.

[132] P. E. Lehner, R. Vane, and K. B. Laskey, "Merging AI and Game Theory in Multiagent Planning," in *Proc. IEEE International Symposium on Intelligent Control*, Philadelphia, PA, 5-7 September 1990, pp. 853-857.

[133] A. Giarmanno, Personal Communication (regarding United States Secret Service use of simulation in training), Special Agent, United States Secret Service, January - February 1996.

[134] D. E. Sackett, "Simulations to Save Time, Money, and Lives," *Lawrence Livermore National Laboratory Science and Technology Review*, vol. 96, no. 11, November 1996, pp. 4-11.

[135] J. Baxter and R. Hepplewhite, "Agents in Tank Battle Simulations," *Communications of the ACM*, vol. 42, no. 3, March 1999, pp. 74-75.

[136] J. Surdu, D. Ragsdale, B. Cox, J. Yen, and U. Pooch, "Implementing Entities in Simulation as Intelligent Agents," in *Proc. Military, Government, and Aerospace 1998*, Boston, MA, 5-9 April 1998, pp. 90-95.

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

[138] X. Lou and J. Su, "A Fuzz Decision Model for Command and Control Process," in *First International Symposium on Uncertainty, Modeling, and Analysis*, University of Maryland, Baltimore, MD, 3 - 5 December 1990, pp. 318 - 320.

[139] J. D. Parsons, "Using Fuzzy Logic Control Technology to Simulate Human Decision-Making in Warfare Models," Tech. Rep. 93-1, Center for Naval Analyses, Alexandria, VA, January 1993.

[140] B. Hayes-Roth, "An Architecture for Adaptive Intelligent Systems," *Artificial Intelligence*, vol. 72, no. 1, January 1995, pp. 329-365.

[141] J. S. Milton and J. C. Arnold, *Introduction to Probability and Statistics: Principles and Applications for Engineering and the Computing Sciences*, New York: McGraw-Hill, Inc., 1995.

[142] U. S. Army, *FM 100-5: Operations*, Washington, DC: Headquarters, Department of the Army, 1993.

[143] D. W. Seidel, "Aggregate Level Simulation Protocol (ALSP) Program Status and History," Tech. Rep. 92-1, MITRE Corporation, McLean, VA, March 1993.

[144] National Imagery and Mapping Agency, "NIMA Public Web Page," Available at http://www.nima.mil, December 1999.

[145] Department of Defense, *MIL-STD-2407: Department of Defense Interface Standard for Vector Product Format*, Fairfax, VA: National Imagery and Mapping Agency, 1996.

[146] National Imagery and Mapping Agency, "VPF Overview Web Page," Available at http://164.214.2.59/vpfproto/index.htm, December 1999.

[147] Department of Defense, *MIL-V-89032: Military Specification: Vector Smart Map (VMap) Level 2 (Draft)*, Fairfax, VA: National Imagery and Mapping Agency, 1996.

[148] Defense Modeling and Simulation Office, "SEDRIS Web Page," Available at http://www.dmso.mil/portals/sedris.html, December 1999.

[149] G. W. Evans, G. F. Wallace, and G. L. Sutherland, *Simulation Using Digital Computers*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1967.

APPENDIX A

MESSAGE PROTOCOL

The Following messages are used between OpSim and any clients who connect to OpSim.

Messages used by a client to subscribe or unsubscribe to information:

#SUBRATE <int seconds>)	sets the rate at which updates are sent to the client
# SUBTIME	subscribes to clock updates
#NOSUBTIME	unsubscribe to clock updates
#SUBLOC	subscribe to location information for all entities
#NOSUBLOC	unsubscribe to location information
#SUBSTAT	subscribe to status information for all entities
#NOSUBSTAT	unsubscribe to status information
#SUBFULL	subscribe to full information updates for all entities
#NOSUBFULL	unsubscribe to full information updates
#SUBSIMSTATUS	subscribe to the status of the simulation
#NOSUBSIMSTATUS	unsubscribe to the status of the simulation

#QUERY RUNNINGPLAN <int sideNum>

asks which plan is currently running for side sideNum

#QUERY XBAR <int bluePlanNum><int redPlanNum>

asks what the mean losses were for the indicated plan matchup

#QUERY RESULTS <className>

asks for the performance results for a certain type of entity. This is used to determine whether the probability distribution used in the simulation matches the real world.

#QUERY AVGDURATION <int bluePlanNum> <int redPlanNum>

asks for the average length of time for the blue plan to run against the red plan before the end of the operation was reached

#QUERY ENTITYLIST asks for the entire entity list

#QUERY SETDISTRIBUTION <className> <distName> <param1> <param2>

lets a client tell OpSim to change the distribution describing the performance of className entities to distName with parameters param1 and param2. If the distribution only uses one parameter, the param2 is ignored, but some value must be present in the message.

The following messages are used by a controlling client (e.g., the GUI) to modify the behavior of the simualtion.

#GO	sets the simulation to start state and begins execution
#PAUSE	pauses the simulation but does not reset the state
#RESUME	continues execution of the simulation from the point at which it was paused
#STOP	terminates execution of the simulation
#MODE <int modeNum>	sets the clock mode to one of those defined in SimClock
#RANDOM <int random>	sets the OurRandom random number generator class to one of the modes defined in OurRandom
#ITERATIONS <int num>	tell the simulation to run num iterations of each plan
#SIMPLAN <sideNum> <planNum>	tells the simulation to run planNum for sideNum against all enemy planNums
#NOSIMPLAN <sideNum> <planNum>	tells the simulation NOT to run planNum for sideNum against all enemy planNums
#COVERAGE <int covNum>	tells the simulation to use the indicated coverage from the terrain database
#NOCOVERAGE <int covNum>	

tells the simulation NOT to use the indicated coverage from the terrain database

#ENTITY … sends all the information about an entity. The content of the rest of the message is determined by the entity. OpSim will use the Java reflection API to determine the correct entity class upon receipt.

#NODE <int entityID> <int planNum> <int nodeID> …

sends all the information about a plan node for entity entityID and plan planNum. The rest of the message is determined by the Node class

#LOCUPDATE <int idNum> <double longitude> <double latitude>
<double elevation>

sends the new longitude, latitude, and elevation (above ground level) for entity idNum

#STATUSUPDATE <int idNum> <int status>

sends the new status of entity idNum

#FULLUPDATE <string …>

sends all state information about an entity. The rest of the information in the message is determined by the entity's class. OpSim uses the Java reflection API to determine the correct class.

#ENTITYLIST <string …>	sends the entire entity list. The rest of the message is exactly the same as what would be output to a scenario file, except the end-of-line characters have been replaced with '|' symbols
#ENTITY <string …>	send all the information about a new entity. This is used to add new entities to the simulation.
#SIMSTATUS <string status>	sends the current status of the simulation: STOPPED, RUNNING, PAUSED, or FINISHED

APPENDIX B

SOURCE CODE

All the source code for the OpSim prototype as well as the Operations Monitors is provided on the accompanying compact disk. The compact disk was written as a standard computer data disk, so it should be readable on any computer. Any utilities written specifically for this research are also included. All of the code was written in standard Java, compliant with version 2.0 of the language. This code can be read in any text editor or Java development environment. The discussion of how to compile and run OpSim and the Operations Monitors assumes familiarity with Java and signed applets.

According to standard Java coding practice, there is a Classes directory on the CD. Within this directory are directories for each of the packages that make up the OpSim prototype. Copy the entire source tree to a readable and writable file system. Set your Java CLASSPATH to this Classes directory. A short script file was created, updateOpSim, for compiling the source tree, creating a jar file, and finally making the jar file a signed jar file. The user will need to create a keystore in order to create the signed applets. The user is referred to Sun's Java Web site for information on how to create a keystore. Once created, the proper information should merely be substituted into the following line of updateOpSim:

```
jarsigner –keystore <store name> -storepass <store password> -keypass
<key password> -signedjar <name of signed jar file to be created> <name
of existing unsigned jar file> signFiles
```

To run OpSim takes three steps. First a Web server must be running on the machine on which the simulation sever will eventually be running. The Acme light Web server is included on this CD. If the user's CLASSPATH has been set to the Classes directory, the Web server can be instantiated with the following command (in Unix):

```
java Acme.Serve.Serve &
```

Then the simulation server must be created on the same machine. This can be done with the following command:

```
java Network.Server <portNum> <dataPath>
```

If portNum and dataPath are omitted, the default port, 8000, is used, and the default data directory of ./Data/Sim will also be used. When a copy of OpSim was used to simulate the real world, the Web server and simulation server were launched on the "simulated" machine as described above, and they were launched again on the "real" machine, with the dataPath set to ./Data/Real. Note that if the user needs to specify a dataPath other than the default, the portNum must also be specified.

Once the Web server and simulation server are running, the user can launch the GUI with the following command:

```
appletviewer –J-Djava.security.policy=Write.jp
   http://<hostname>:9090/AppletClient.html
```

As with all the other commands, this one should be executed from within the "root" directory of the source code on the CD. The applet is looking for data in ./Data/Sim or ./Data/Real. A sample security policy file, Write.jp, is included with the source. Write.jp can be modified by the user to reflect the signing users of OpSim in the

keystore. Hostname is the name of the machine on which the Web and simulation servers are running. On a non-networked PC, *localhost* can be substituted.

On the applet GUI there are six text boxes to be filled in and a button, labeled "Connect." Fill in the text boxes with the following information:

Hostname of Server	The name of the machine on which the Web and simulation servers are running
Port of Sim Server	The port number on which OpSim will be listening for command connections.
Port of Query Server	The port number on which OpSim will be listening for subscription connections.
Random Number Seed	If 0 is typed in this box, a random number seed will be automatically generated. For purposes of repeatability, the user can specify an integer seed value.
Data Path	The path to data files from the directory in which the appletviewer command was typed. Usually one OpSim looks in Data/Sim and another looks in Data/Real
Choose Terrain	Not implemented!

After a simulation is running both on a "simulated" and on a "real" machine, the user can launch the top-level Operations Monitor with the following command:

```
java Monitors.TopLevelMonitor –f Monitors.setup
```

This command tells the TopLevelMonitor to read its parameters from the file Monitors.setup. Each of these parameters can also be typed on the command line.

In addition to the source code and compiled class files, in the Docs directory is the documentation created by using the javadoc utility with all private and protected methods shown. This will be helpful to any future developer of OpSim.

VITA

John R. Surdu was born 3 July 1963 in Ypsilanti, Michigan. He is a 1985 graduate of the United States Military Academy, West Point, where he earned the Bachelor of Science degree in Computer Science and was commissioned as an infantry officer in the U.S. Army. He has served in a number of positions in the Army, achieving the rank of Major. During that period he earned the Master of Business Administration degree from Columbus State University in 1992 and the Master of Science degree in Computer Science from Florida State University in 1995. In addition to more traditional military assignments, he served at the Army Research Laboratory where he worked with several military constructive simulations and was team leader for the Virtual Sand Table project. In 2000 he was awarded a doctoral degree in Computer Science at Texas A&M University in preparation for a faculty position at West Point.

John R. Surdu's permanent address is 111 Eastland, Charles Town, West Virginia, 25414.